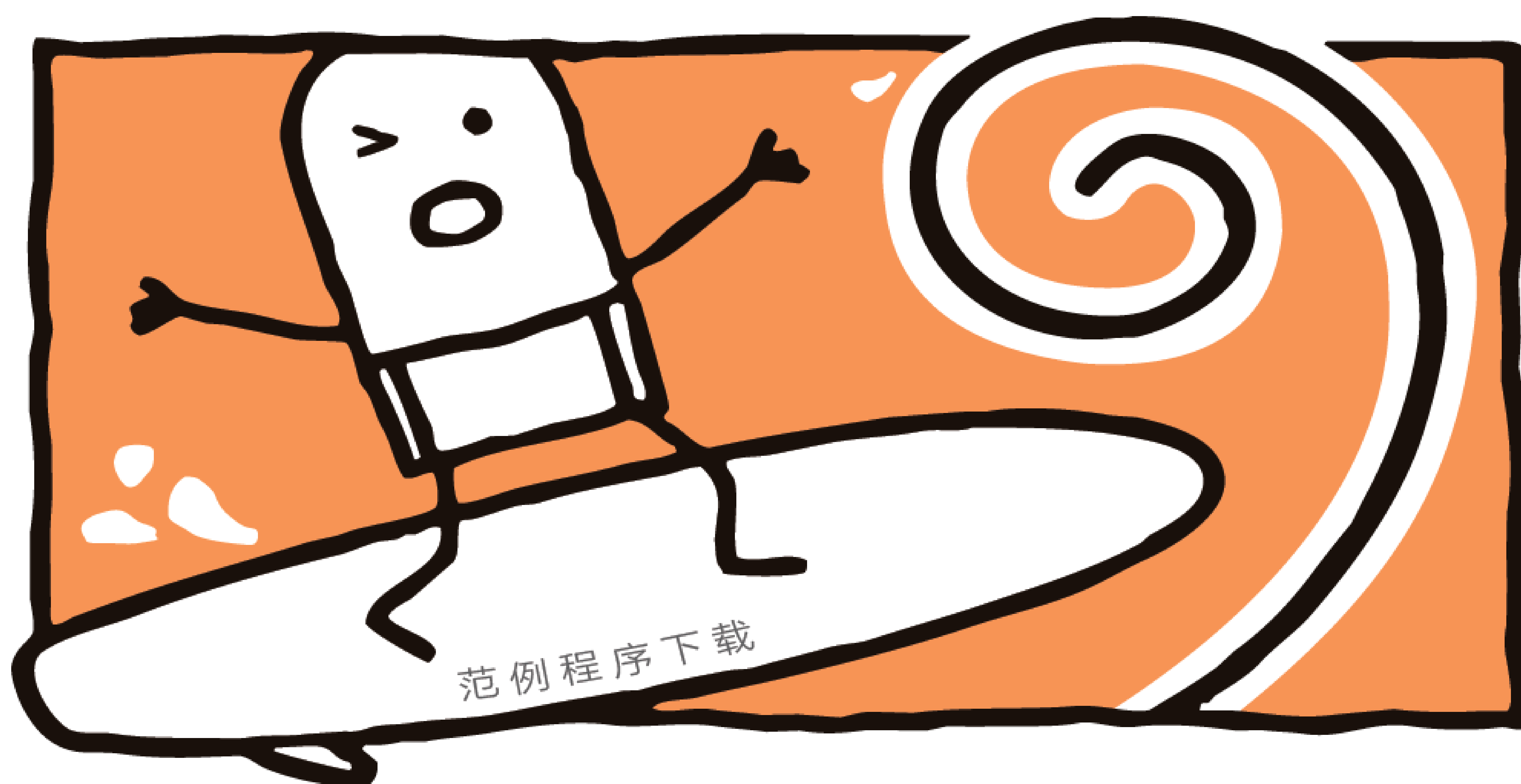


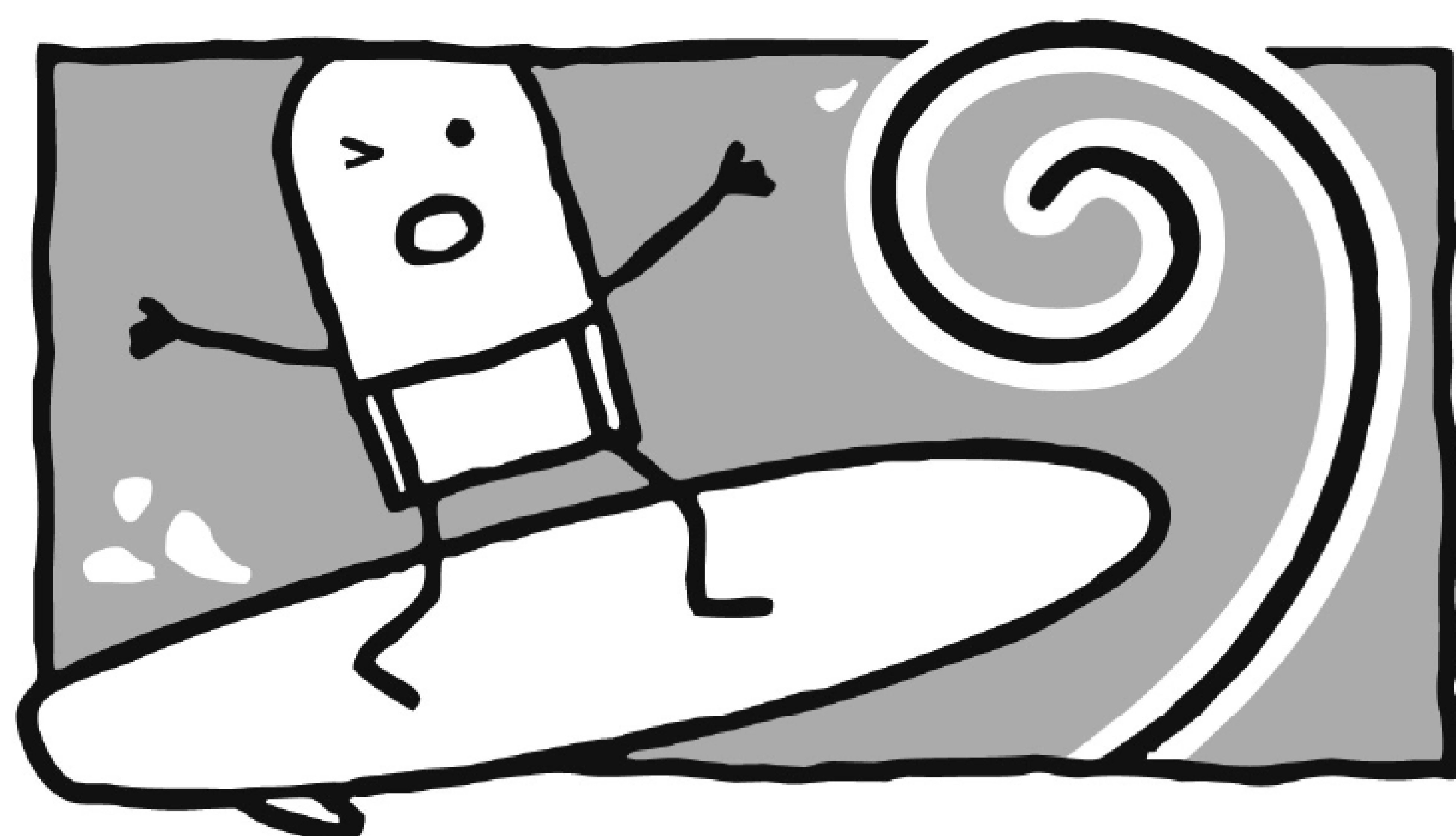
- 浅显易懂的原理介绍
- Step by Step 实机操作、范例程序详细解说
- 降低机器学习与大数据技术的学习门槛

Python+Spark 2.0+Hadoop 机器学习与大数据实战

林大贵 著

轻松快速学会机器学习与大数据热门技术 

清华大学出版社



Python+ Spark 2.0+Hadoop 机器学习与大数据实战

林大贵 著

清華大學出版社
北京

内 容 简 介

本书从浅显易懂的“大数据和机器学习”原理说明入手，讲述大数据和机器学习的基本概念，如分类、分析、训练、建模、预测、机器学习（推荐引擎）、机器学习（二元分类）、机器学习（多元分类）、机器学习（回归分析）和数据可视化应用等。书中不仅加入了新近的大数据技术，还丰富了“机器学习”内容。

为降低读者学习大数据技术的门槛，书中提供了丰富的上机实践操作和范例程序详解，展示了如何在单机Windows 系统上通过 Virtual Box 虚拟机安装多机 Linux 虚拟机，如何建立 Hadoop 集群，再建立 Spark 开发环境。书中介绍搭建的上机实践平台并不限制于单台实体计算机。对于有条件的公司和学校，参照书中介绍的搭建过程，同样可以实现将自己的平台搭建在多台实体计算机上，以便更加接近于大数据和机器学习真实的运行环境。

本书非常适合于学习大数据基础知识的初学者阅读，更适合正在学习大数据理论和技术的人员作为上机实践用的教材。

图书在版编目（CIP）数据

Python+Spark 2.0+Hadoop 机器学习与大数据实战 / 林大贵著.—北京：清华大学出版社，2018
ISBN 978-7-302-49073-9

Ⅰ. ①P… Ⅱ. ①林… Ⅲ. ①软件工具－程序设计②数据处理软件 Ⅳ. ①TP311.561②TP274

中国版本图书馆 CIP 数据核字（2017）第 296017 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：李红英
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**33.75　　**字　　数：**864 千字
版　　次：2018 年 1 月第 1 版　　**印　　次：**2018 年 1 月第 1 次印刷
印　　数：1~3000
定　　价：99.00 元

产品编号：073908-01

序

本书将方兴未艾的“机器学习”和热门的“大数据分析”技术与应用在一本书中融会贯通地娓娓道来，体现了作者深厚的技术功底和丰富的经验。和已经出版的《Hadoop+Spark 大数据巨量分析与机器学习整合开发实战》一书相比，本书不是简单的更新和升级，而是在原有的基础上增加了大数据技术，还丰富了其中略显薄弱的“机器学习”内容，增加了 4 章都和机器学习有关的内容。另外，作者还用流行的“胶水语言”Python 重新改写了另一本书中的范例程序，并添加了“机器学习”+“大数据”章节的范例程序，所以将书名改为“Python + Spark 2.0 + Hadoop 机器学习与大数据实战”，更加突出“机器学习”，并且强调范例程序是运用更加流行的 Python 语言来编写的。

在因特网、社交媒体、电子商务等交叉发展和呼应下，“网络”这个巨人已经拥有了难以计数的海量数据，虽有传统结构化的数据、半结构化的数据，但更多的是非结构化的数据。这些貌似杂乱无章、毫无意义的海量数据是一座等待发掘的巨大“金矿”。这些海量数据中蕴含着极为丰富的人类知识库，是一笔巨大的信息资产。随着云计算时代的来临，对这些原本很难收集整理的大数据进行及时甚至是实时分析和处理并加以有效利用就不再是“海市蜃楼”了。

与大数据相关的内容不外乎三方面：大数据理论，大数据分析和处理的技术（机器学习为核心技术），大数据的实践应用。在与大数据有关的出版物中，偏重于理论教学和技术介绍一类的比较多，而偏重于上机实践和自学的书比较少见。因此，本书非常适合“机器学习和大数据分析”的初学者和正在学习这个领域技术的人员作为学习和上机实践用的教材。

本书不是对原理进行纯理论的阐述，而是提供了丰富的上机实践操作和范例程序，从而降低了读者学习“机器学习和大数据分析”的门槛。对于需要直接上机实践的学习者而言，本书更像是一本学习实践和实战开发的上机手册。书中首先展示了如何在单台 Windows 系统上通过 Virtual Box 虚拟机安装多台 Linux 虚拟机，而后建立 Hadoop 集群，再建立 Spark 开发环境。搭建这个上机实践的平台并不限制于单台实体计算机，主要是考虑个人读者上机实践的实际条件和环境。对于有条件的公司和学校，参照这个搭建过程，同样可以将实践平台搭建在多台实体计算机上。另外，现在很多大专院校都开设了 Python 程序设计语言的课程，所以本书的所有范例程序都用 Python 语言重新改写了，非常接“地气”。

在搭建好“机器学习和大数据分析”上机实践的软硬件环境之后，就可以在各章节的学习中结合本书提供的范例程序逐一设置、修改、调试和运行，从中学到“机器学习和大数据分析”

实践应用中核心技术的真谛——对大数据进行高效的“智能加工”，萃取大数据中蕴含的“智慧和知识”，实现数据的“增值”，并最终将其应用于实际工作或者商业中。

对于企业在商业应用中的“机器学习和大数据分析”核心系统，需要运用商业公司的解决方案作为引擎。在中国市场活跃的国际和国内著名公司也提供了相当好的解决方案，比如 Cloudera 对 Spark ml 提供完整的支持、星环科技基于 Spark 自主研发了机器学习产品 Discover。

大数据与云计算的关系密不可分，涉及众多关键技术，比如分布式处理、分布式数据库和云存储、虚拟化技术等，但是它们不是本书的重点，所以这里并未深入讲解。建议需要深入学习这方面内容的读者去寻找相关出版物，结合本书的实践来丰富和完善自己的大数据知识体系。

资深架构师 赵军

2017 年 11 月

前言

机器学习是近二十来年兴起的多领域学科，机器学习算法可从数据中建立模型，并利用模型对未知数据进行预测。机器学习技术不断进步，应用相当广泛，例如推荐引擎、定向广告、需求预测、垃圾邮件过滤、医学诊断、自然语言处理、搜索引擎、诈骗侦测、证券分析、视觉辨识、语音识别、手写识别等。

近年来 Google、Facebook、Microsoft、IBM 等大公司全力投入机器学习研究与应用。以 Google 为例，Google 已经将机器学习运用到垃圾邮件判断、自动回复、照片分类与搜索、翻译、语音识别等功能上。同时，各大主流 Hadoop 发行版公司加强了对机器学习的投入，比如 Cloudera 对 spark ml 的完整支持、星环科技基于 Spark 自主研发的机器学习产品 Discover。在不知不觉中，机器学习已经让日常生活更为便利。

为什么近年来机器学习变得如此热门，各大公司都争相投入？因为机器学习需要大量数据进行训练。大数据的兴起带来了大量的数据以及可存储大量数据的分布式存储技术，例如 Hadoop HDFS、NoSQL……还有分布式计算可进行大量运算，例如 Spark 基于内存的分布式计算框架/架构，可以大幅提升性能。

本书的主题是 Python+Spark+Hadoop 机器学习与大数据分析。使用 Python 开发 Spark 应用程序，具有多重优势：不仅可以享有 Python 语言特性所带来的好处，即程序代码简明、较易学习、高生产力等，再加上 Spark 基于内存的分布式计算框架/架构，还可以大幅提升性能，非常适合需要多次重复运算的机器学习算法，并且 Spark 还可以存取 Hadoop HDFS 分布式存储的大量数据。

本书希望能够用浅显易懂的原理介绍和说明以及上机实践操作、范例程序来降低机器学习与大数据技术的学习门槛，带领读者进入机器学习和大数据的领域。当然，整个机器学习与大数据的生态系统非常庞大，需要学习的东西很多。读者通过本书学习，对机器学习和数据有了基本的概念后就比较容易踏入这个领域了，以便深入研究其他的相关技术。

林大贵

本书章节与范例程序介绍

本书特色

提供了大量上机实践操作与范例程序。

➢ **上机实践操作**

一般人可能会认为机器学习和大数据分析需要很多台机器的环境才能学习，实际上通过本书使用 Virtual Box 虚拟机的方法就能在自家的计算机上演练建立 Hadoop 集群以及 Python Spark 开发环境。同时，上机实践操作介绍了 Hadoop MapReduce 与 HDFS 的基本概念，以及 Spark RDD、DataFrame、Spark SQL 与 MapReduce 的基本概念。

➢ **范例程序**

以实际范例程序来学习程序设计是最有效率的学习方式，因此本书使用实际的数据集，配合范例程序代码来介绍各种机器学习的算法，并示范如何获取数据、训练数据、建立模型、预测结果，由浅入深地介绍 Python Spark 机器学习。

本书章节内容及上机实践操作与范例程序介绍

➢ **基本概念介绍**

章节	章节名称	说明
1	Python Spark 机器学习与 Hadoop 大数据	介绍机器学习、Spark 基本概念、Python 开发 Spark 机器学习与大数据应用、Spark ML Pipeline 机器学习流程、大数据定义、Hadoop 基本概念、HDFS、MapReduce 等基本原理

➢ **Hadoop 的安装**

章节	章节名称	说明
2	Virtual Box 虚拟机软件的安装	**上机实践操作** 安装 Virtual Box 虚拟机，让你可以在 Windows 系统上安装多台 Linux 虚拟机
3	Ubuntu Linux 操作系统的安装	**上机实践操作** 在 Virtual Box 虚拟机上安装 Ubuntu Linux 操作系统

（续表）

章节	章节名称	说明
4	Hadoop Single Node Cluster 的安装	**上机实践操作** 在 Ubuntu Linux 的操作系统上安装单台机器的 Hadoop Single Node Cluster
5	Hadoop Multi Node Cluster 的安装	**上机实践操作** 在 Ubuntu Linux 的操作系统上安装多台机器 Hadoop Multi Node Cluster，并介绍 Hadoop Resource-Manager 与 NameNode HDFS Web 界面

➢ **Hadoop 的基本功能**

章节	章节名称	说明
6	Hadoop HDFS 命令	**上机实践操作** 示范如何使用 HDFS 命令，并介绍 Hadoop HDFS Web 界面
7	Hadoop MapReduce	WordCount.java 范例程序 介绍 Hadoop MapReduce 原理，示范如何使用 Hadoop MapReduce 计算文章内的每一个单词（或字）出现的次数

➢ **Spark 的基本功能介绍**

章节	章节名称	说明
8	Python Spark 的安装与介绍	**上机实践操作** 示范如何安装 Python Spark，并在 pyspark“终端”程序界面中在本地以 Hadoop YARN-client 或 Spark Stand Alone 模式来运行 Python Spark 程序
9	在 IPython Notebook 运行 Python Spark 程序	**上机实践操作** 示范如何安装 Aanconda Python 软件包，在 IPython Notebook 中在本地以 Hadoop YARN-client 或 Spark Stand Alone 模式来运行 Python Spark 程序
10	Python Spark RDD	**上机实践操作** Spark 基本功能 RDD（Resilient Distributed Dataset，弹性分布式数据集）的基本运算
11	Python Sparkr 的集成开发环境	**上机实践操作** 示范如何安装 eclipse+pyDev 集成开发环境来运行 Python Spark 程序 WordCount.py 范例程序 示范如何在 eclipse 中在本地以 Hadoop YARN-client 或 Spark Stand Alone 模式运行 Python Spark 程序

➢ **以 RDD 为基础的 Spark MLlib 机器学习**

章节	章节名称	说明
12	Python Spark 创建推荐引擎	IPython Notebook 范例程序 通过 IPython Notebook 交互式界面，示范如何使用 Spark MLlib 命令建立电影的推荐引擎 RecommendTrain.py 范例程序 示范如何提取数据、训练并建立模型、存储模型 Recommend.py 范例程序 示范如何载入模型、使用模型推荐用户或电影
13	Python Spark MLlib 决策树二元分类	RunDecisionTreeBinary.py 范例程序 示范如何使用决策树二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性或可以长久存在的，并且找出最佳参数组合，提高预测准确度
14	Python Spark MLlib 逻辑回归二元分类	RunLogisticRegressionWithSGDBinary.py 范例程序 示范如何使用逻辑回归二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性或可以长久存在的，并找出最佳参数组合，提高预测准确度
15	Python Spark MLlib 支持向量机 SVM 二元分类	RunSVMWithSGDBinary.py 范例程序 示范如何使用支持向量机 SVM 二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性或可以长久存在的，并找出最佳参数组合，提高预测准确度
16	Python Spark MLlib 朴素贝叶斯二元分类	RunNaiveBayesBinary.py 范例程序 示范如何使用朴素贝叶斯二元分类分析 StumbleUpon 数据集，预测哪些网页是暂时性或可以长久存在的，并找出最佳参数组合，提高预测准确度
17	Python Spark MLlib 决策树多元分类	RunDecisionTreeMulti.py 范例程序 示范如何使用决策树多元分类分析 Covertype 数据集（森林覆盖植被），根据不同的土地条件可以预测该地的植被，并找出最佳参数组合，提高预测准确度
18	Python Spark MLlib 决策树回归分析	RunDecisionTreeRegression.py 范例程序 示范如何使用决策树回归分析 Bike Sharing（共享单车）数据集。根据天气假日条件，可以预测每一个小时租借的数量，并找出最佳参数组合，提高预测准确度

➢ **以 DataFrame 为基础的 Spark ML Pipeline 机器学习流程**

章节	章节名称	说明
19	Python Spark SQL、DataFrame、RDD 数据统计与可视化	IPython Notebook 范例程序 通过 IPython Notebook 交互式界面介绍并比较 Spark 数据的处理方式：DataFrame vs Spark SQL vs RDD，并且使用 Pandas 与 matplotlib 绘图
20	Spark ML Pipeline 机器学习流程二元分类	IPython Notebook 范例程序 以“StumbleUpon”数据集示范如何使用 Spark ML Pipeline 机器学习流程二元分类，预测网页是暂时性的还是长久存在的，并且使用训练验证与交叉验证找出最佳模型，提高预测准确度，最后介绍如何使用随机森林 RandomForestClassifier 分类算法进一步提高准确率
21	Spark ML Pipeline 机器学习流程多元分类	IPython Notebook 范例程序 以“森林覆盖植被”多元分类数据集示范如何使用 Spark ML Pipeline 机器学习流程多元分类，预测 Cover Type 森林覆盖分类，并且使用训练验证与交叉验证找出最佳模型，提高预测准确度
22	Spark ML Pipeline 机器学习流程回归分析	IPython Notebook 范例程序 以“Bike Sharing”数据集示范如何使用 Spark ML Pipeline 机器学习流程回归分析，预测每一小时租借总数量，并且使用训练验证与交叉验证找出最佳模型，提高预测准确度，最后介绍使用 GBT（Gradient-Boosted Trees，梯度提升决策树）进一步提高预测准确度

本书范例程序下载与安装说明

可参考附录 A 中有关本书范例程序下载与安装的说明。本书范例程序主要分为两个部分：

范例	说明
IPython Notebook 范例	第 9、10、12、13、19、20、21、22 章。按照第 9 章的说明来安装 anaconda 及其设置后才能使用这些范例程序
eclipse 范例	第 11~18 章。按照第 11 章的说明完成 eclipse 的安装与全部设置后才能执行这些范例程序

本书上机实践操作命令的整理

本书第 2 章到第 10 章使用了很多 Linux、spark-shell、SparkSQL 等命令。不过很多命令都很长，只要有一个字母打错就无法运行，这样会增加挫折感。因此我们在博客文章中整理了各个章节使用的命令，可参考如下网页：

http://www.weibo.com/hadoopsparkbook

安装或练习命令时，你可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可以节省打字的时间，又不用担心打错字母（无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，可参考第 3.9 节的说明设置好 VirtualBox 的共享剪贴板）。

读者服务与社区交流

在网络时代，购买本书的读者不仅可以获得本书的内容，还能通过网络社区获得更多的信息。

➢ **本书的博客**

网址：http://blog.sina.com.cn/hadoopsparkbook。

我们将一些需要排列整齐、系统化的信息放在了博客文章中，还会随时更新，内容包括：

- 本书上机实践操作命令的整理。
- 本书内容或程序代码的勘误。
- 分享最新的 Hadoop 或 Spark 信息。

➢ **本书的微博**

网址：http://www.weibo.com/hadoopsparkbook。

我们建立了本书的 Facebook 粉丝团，欢迎读者们加入。粉丝团会不定期贴文，分享最新的 Hadoop 或 Spark 信息，大家可以随时提问并参与交流。

➢ **百度网盘**

网址：**http://pan.baidu.com/s/1i4AzAk9**（注意区分数字和英文字母大小写）

如果下载有问题，请发送电子邮件至 booksaga@126.com，邮件主题设置为“求 Python+Spark 2.0+Hadoop 机器学习与大数据实战范例程序”。

目　录

第1章

Python Spark机器学习与Hadoop大数据

本章将介绍机器学习、Spark 基本概念、使用 Python 开发 Spark 机器学习与大数据应用、Spark ML Pipeline 机器学习流程、大数据定义、Hadoop 基本概念、HDFS、MapReduce 等基本原理。

1.1 机器学习的介绍

机器学习技术不断进步，应用相当广泛，例如推荐引擎、定向广告、需求预测、垃圾邮件过滤、医学诊断、自然语言处理、搜索引擎、欺诈检测、证券分析、视觉识别、语音识别、手写识别等。

➢ 机器学习架构

机器学习（Machine Learning）通过算法、使用历史数据进行训练，训练完成后会产生模型。未来当有新的数据提供时，我们可以使用训练产生的模型进行预测。

机器学习训练用的数据是由 **Feature**、**Label** 组成的。

- **Feature**：数据的特征，例如湿度、风向、风速、季节、气压。
- **Label**：数据的标签，也就是我们希望预测的目标，例如降雨（0.不会下雨、1.会下雨）、天气（1.晴天、2.雨天、3.阴天、4.下雪）、气温。

如图 1-1 所示，机器学习可分为以下两个阶段。

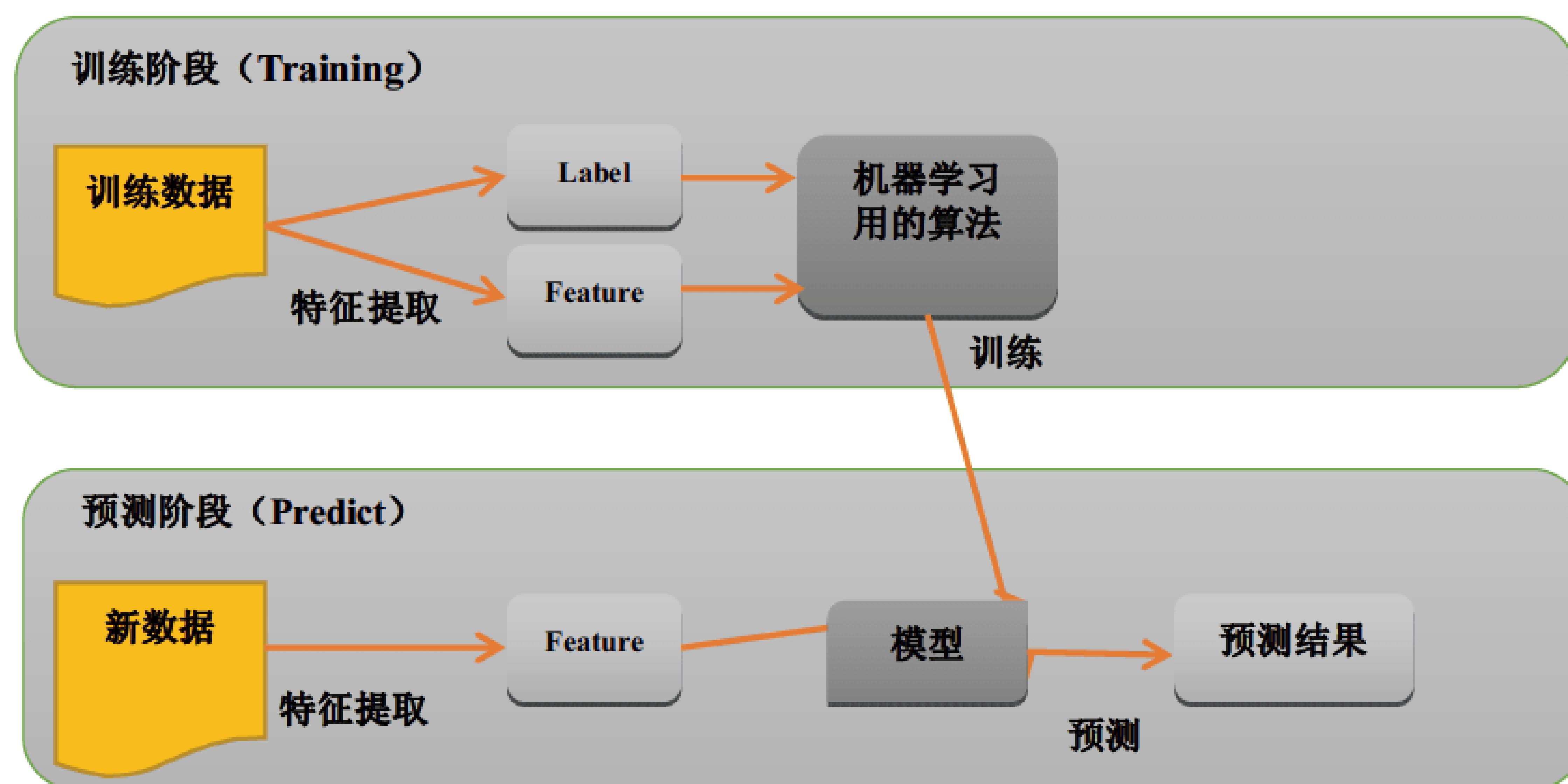


图 1-1　机器学习的两个阶段

- **训练阶段**（Training）

训练数据是过去累积的历史数据，可能是文本文件、数据库或其他来源。经过 Feature Extraction（特征提取），产生 Feature（数据特征）与 Label（预测目标），然后经过机器学习算法的训练后产生模型。

- **预测阶段**（Predict）

新输入数据，经过 Feature Extraction（特征提取）产生 Feature（数据特征），使用训练完成的模型进行预测，最后产生预测结果。

➢ 机器学习分类

对于有监督的学习（Supervised Learning），从现有数据我们希望预测的答案有下列分类。

- 二元分类

我们已知湿度、风向、风速、季节、气压等数据特征，希望预测当天是否会下雨（0. 不会下雨、1. 会下雨）。因为希望预测的目标 Label 只有 2 种选项，所以就像是非题。

- 多元分类

我们已知湿度、风向、风速、季节、气压等数据特征，希望预测当天的天气（1. 晴天、2. 雨天、3. 阴天、4. 下雪）。因为希望预测的目标 Label 有多个选项，所以就像选择题。

- 回归分析

我们已知湿度、风向、风速、季节、气压等数据特征，希望预测当天的气温。因为希望预测的目标 Label 是连续值，所以就像是计算题。

对于无监督的学习（Unsupervised Learning），从现有数据我们不知道要预测的答案，所以没有 Label（预测目标）。cluster 聚类分析的目的是将数据分成几个相异性最大的群组，而群组内的相似性最高。

根据上述内容我们可以整理出如表 1-1 所示的内容。

表 1-1　机器学习分类表

分类	算法	Features（特征）	Label（预测目标）
有监督的学习	二元分类 （Binary Classification）	湿度、风向、风速、季节、气压……	只有 0 与 1 选项（是非题） 0. 不会下雨、1. 会下雨
有监督的学习	多元分类 （Multi-Class Classification）	湿度、风向、风速、季节、气压……	有多个选项（选择题） 1. 晴天、2. 雨天、3. 阴天、4. 下雪
有监督的学习	回归分析 （Regression）	湿度、风向、风速、季节、气压……	值是数值（计算题） 温度可能是 -50～50 度的范围
无监督的学习	聚类分析 （Clustering）	湿度、风向、风速、季节、气压……	无 Label Cluster 聚类分析，目的是将数据分成几个相异性最大的群组，而群组内的相似性最高

机器学习分类可以整理成图 1-2。

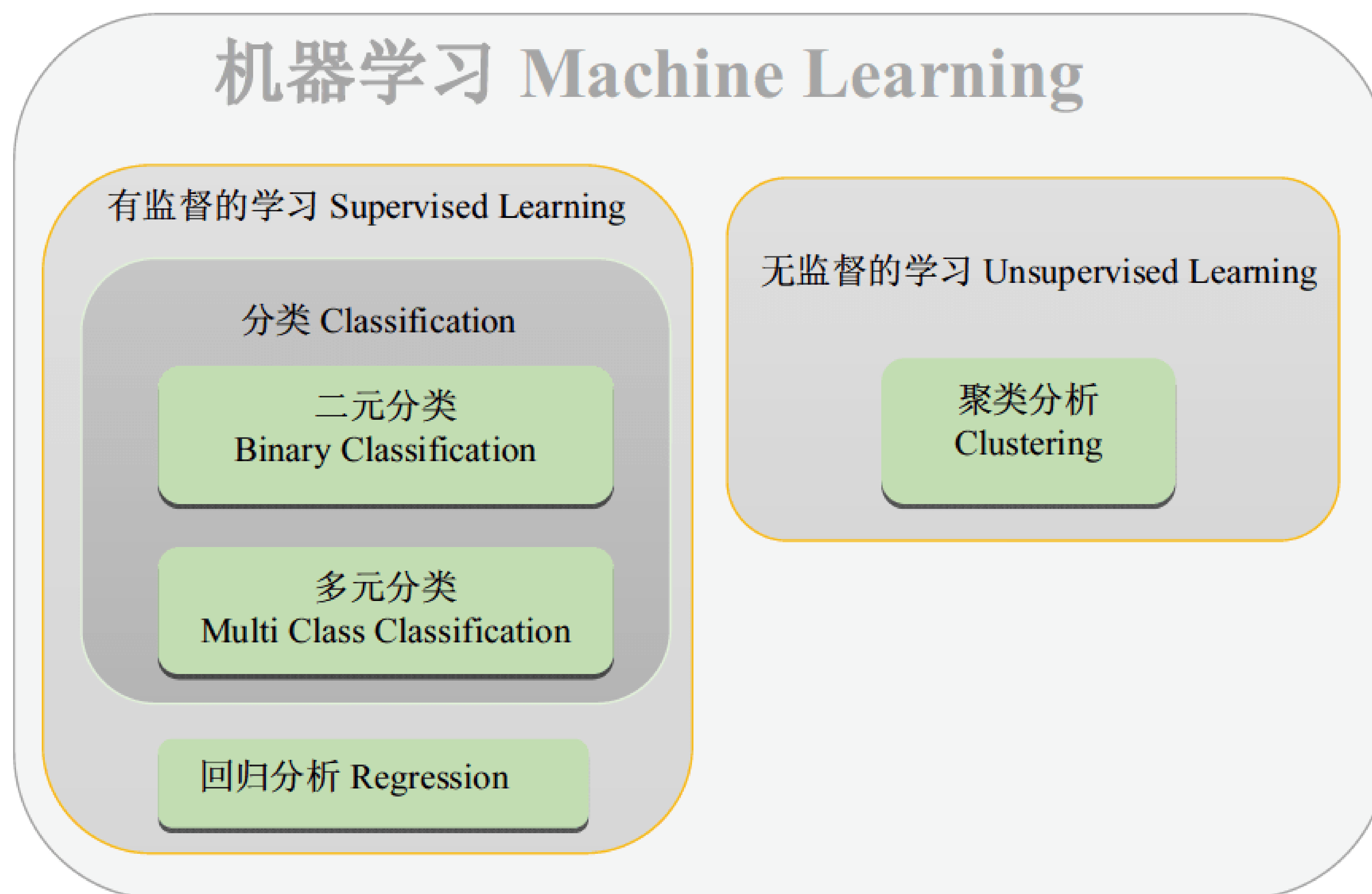


图 1-2　机器学习分类图

如图 1-3 所示，机器学习程序的运行可分为下列 3 个阶段。

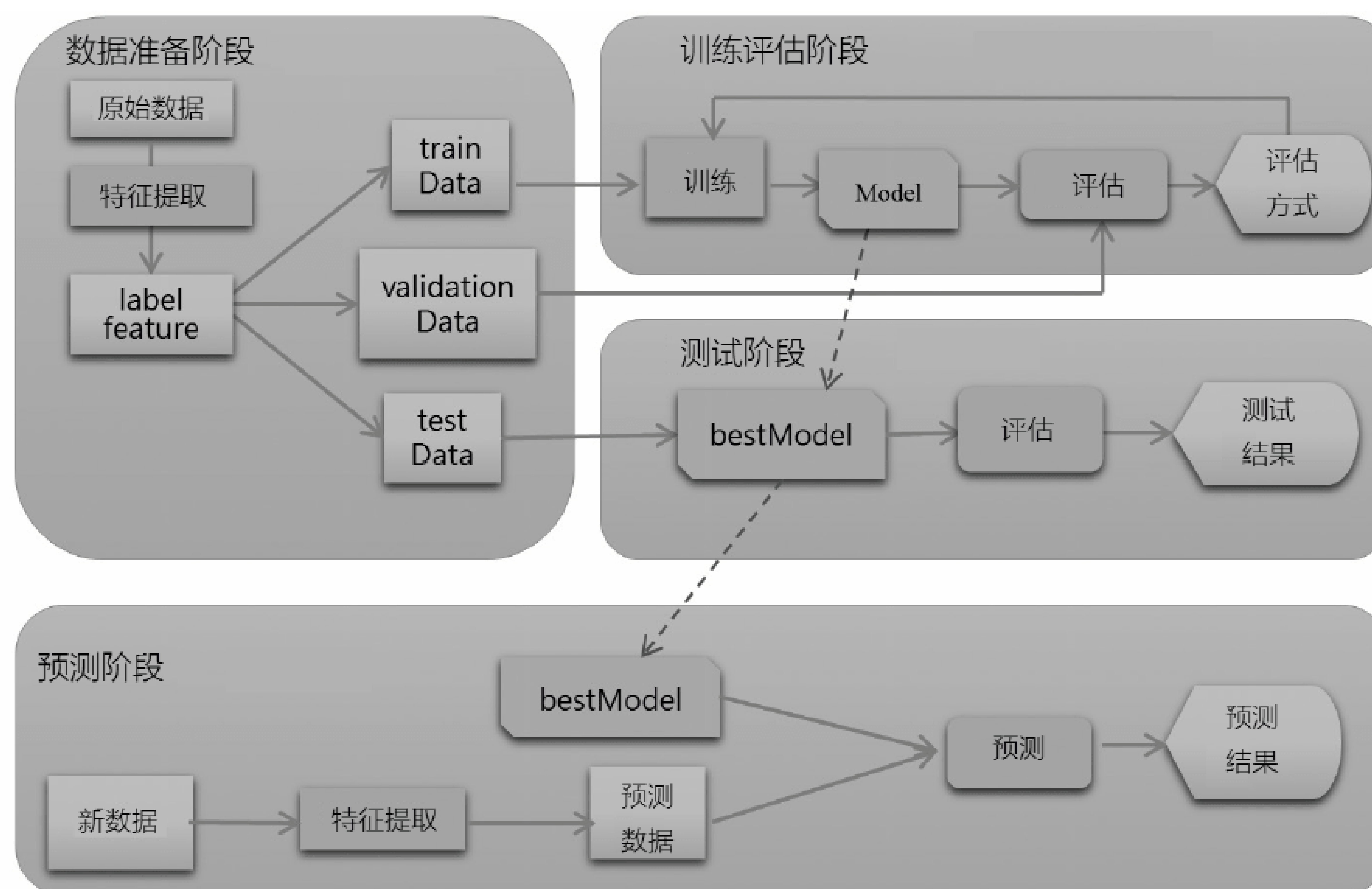


图 1-3　机器学习程序的运行架构

- 数据准备阶段

原始数据（可能是文本文件、数据库或其他来源）经过数据转换，提取特征字段与标签字段，产生机器学习所需要的格式，然后将数据以随机方式分为 3 部分（trainData、validationData、testData）并返回数据，供下一阶段训练评估使用。

- 训练评估阶段

我们将使用 trainData 数据进行训练，并产生模型，然后使用 validationData 验证模型的准确率。这个过程要重复很多次才能够找出最佳的参数组合。评估方式：二元分类使用 AUC、多元分类使用 accuracy、回归分析使用 RMSE。训练评估完成后，会产生最佳模型 bestModel。

- 测试阶段

之前阶段产生了最佳模型 bestModel，我们会使用另外一组数据 testData 再次测试， 以避免 overfitting 的问题。如果训练评估阶段准确度很高，但是测试阶段准确度很低，代表可能有 overfitting 的问题。如果测试与训练评估阶段的结果准确度差异不大，代表无 overfitting 的问题。

- 预测阶段

新输入数据，经过 Feature Extraction（特征提取）产生 Feature（数据特征），使用训练完成的最佳模型 bestModel 进行预测，最后产生预测结果。

1.2 Spark 的介绍

Apache Spark 是开放源码的集群运算框架，由加州大学伯克利分校的 AMPLab 开发。Spark 是一个弹性的运算框架，适合进行 Spark Streaming 数据流处理、Spark SQL 互动分析、MLlib 机器学习等应用，因此 Spark 可作为一个用途广泛的大数据运算平台。Spark 允许用户将数据加载到 cluster 集群的内存中存储，并多次重复运算，非常适合用于机器学习的算法。

➢ Spark RDD in-memory 的计算框架

如图 1-4 所示，Spark 的核心是 RDD（Resilient Distributed Dataset）弹性分布式数据集，是由 AMPLab 实验室所提出的概念，属于一种分布式的内容。Spark 主要的优势来自 RDD 本身的特性，RDD 能与其他系统兼容，可以导入外部存储系统的数据集，例如 HDFS、HBase 或其他 Hadoop 数据源。

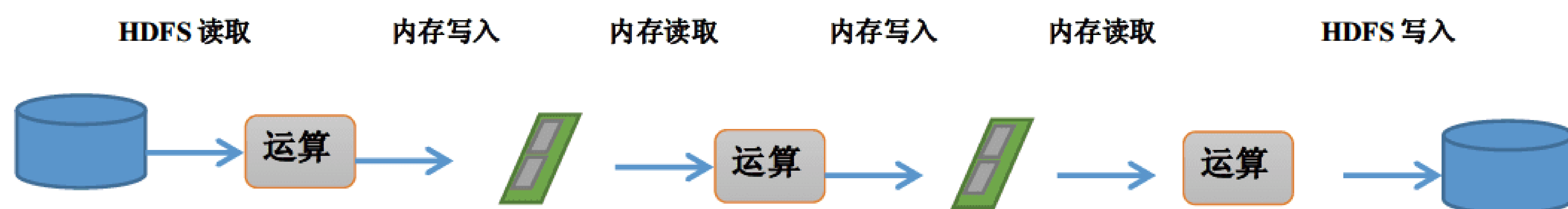


图 1-4 Spark in-memory 的计算框架

Spark 在运算时，将中间产生的数据暂存在内存中，因此可以加快运行速度。需要反复操作的次数越多，所需读取的数据量越大，就越能看出 Spark 的性能。Spark 在内存中运行程序，命令运行速度（或命令周期）比 Hadoop MapReduce 的命令运行速度快 100 倍，即便是运行于硬盘上时 Spark 的速度也能快上 10 倍。

➢ Spark 的发展历史（见表 1-2）

表 1-2 Spark 发展历史

年代	说明
2009	Spark 在 2009 年由 Matei Zaharia 在加州大学柏克利分校的 AMPLab 开发
2010	2010 年通过 BSD 授权条款发布开放源码
2013	该项目被捐赠给 Apache 软件基金会
2014/2	Spark 成为 Apache 的顶级项目
2014/11	Databricks 团队使用 Spark 刷新数据排序的世界纪录
2015/3	Spark 1.3.0 版发布，开始加入 DataFrame 与 Spark ML
2016/7	Spark 2.0.0 版发布，提升执行性能，更容易使用

➢ Spark 的特色（见表 1-3）

表 1-3 Spark 特色

特色	说明
命令周期短	Spark 是基于内存计算的开放源码集群运算系统，比原先的 Hadoop MapReduce 快 100 倍
易于开发程序	目前 Spark 支持多种语言：Scala、Python、Java，也就是说开发者可以根据应用的环境来决定使用哪一种语言开发 Spark 程序，更具弹性，更符合开发时的需求
与 Hadoop 兼容	Spark 提供了 Hadoop Storage API，使它支持 Hadoop 的 HDFS 存储系统，并且支持 Hadoop YARN，可共享存储资源与运算，而且几乎与 Hive 完全兼容
可在各平台运行	• 我们可以在本地端的机器上运行 Spark 程序，只需要 import Spark 的链接库即可 • 也可以在自有的群集上运行 Spark 程序，例如 Spark stand alone、Hadoop YARN 、Mesos 等自行建立的群集 • 针对更大规模的计算工作，我们可以选择将 Spark 程序送至 AWS 的 EC2 平台上运行，按照用户使用的计算资源计费

➢ Spark 2.0 主要功能（参考图 1-5、表 1-4）

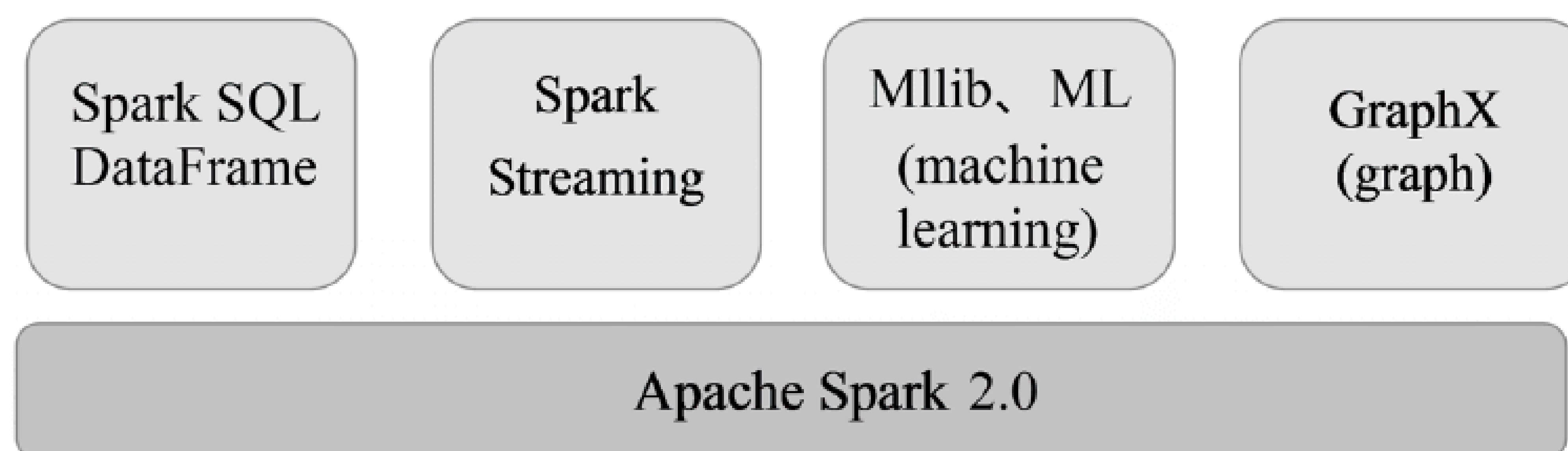


图 1-5 Spark 的主要功能模块

表 1-4　Spark 2.0 主要功能说明

功能	说明
Spark SQL DataFrame	Spark SQL 可以使用熟知的 SQL 查询语言来运行数据分析 DataFrame 具有 Schema（定义字段名与数据类型），可使用类 SQL 方法，例如 select()，使用上比 RDD 更方便
Spark Streaming	Spark Streaming 可实现实时的数据串流的处理，具有大数据量、容错性、可扩充性等特点
GraphX	GraphX 是 Spark 上的分布式图形处理架构，可用图表计算
Spark MLlib Spark ML Pipeline	Spark MLlib 是一个可扩充的 Spark 机器学习库，可使用许多常见的机器学习算法，简化大规模机器学习的时间。算法包括分类与回归、支持向量机、回归、线性回归、决策树、朴素贝叶斯、聚类分析、协同过滤等 Spark ML Pipeline 将机器学习的每一个阶段建立成 Pipeline 流程，可减轻数据分析师程序设计的负担

本书第 8~11 章将介绍 Spark 2.0 的安装与基本功能。

1.3　Spark 数据处理 RDD、DataFrame、Spark SQL

Spark 数据处理方式主要有 3 种：RDD、DataFrame、Spark SQL。

➢ **RDD、DataFrame、Spark SQL 比较**

RDD、Spark DataFrame 与 Spark SQL 最主要的差异在于是否定义 Schema。

- RDD 的数据未定义 Schema（也就是未定义字段名及数据类型）。使用上必须有 Map/Reduce 的概念，需要高级的程序设计能力，但是功能也最强，能完成所有 Spark 功能。
- Spark DataFrame 建立时必须定义 Schema（也就是定义每一个字段名与数据类型），所以 DataFrame 在早期版本中也称为 Schema RDD。
- Spark SQL 是由 DataFrame 衍生出来的，我们必须先建立 DataFrame，然后通过登录 Spark SQL temp table，就可以使用 Spark SQL 语法了。

➢ **易使用度：Spark SQL > DataFrame > RDD**

下面我们以性别统计为例，说明这 3 种方式的难易度。

- 使用 Spark SQL 最简单，只需要使用 SQL 语句即可。即使是非程序设计人员，只要懂得 SQL 语句也可以使用。

```
In [62]: sqlContext.sql("""
         SELECT gender ,count(*)  counts
         FROM   user_table
         GROUP BY gender""").show()

         +------+------+
         |gender|counts|
         +------+------+
         |     F|   273|
         |     M|   670|
         +------+------+
```

- DataFrame API 有 Schema，可直接指定字段名，并且定义了很多类似 SQL 的方法，例如 select()、groupby()、count()，我们可以使用这些方法进行统计，比 RDD 更易使用。只是使用 DataFrame API 必须具有基础程序设计能力。

```
In [64]: user_df.select("gender")    \
                .groupby("gender") \
                .count().show()

         +------+-----+
         |gender|count|
         +------+-----+
         |     F|  273|
         |     M|  670|
         +------+-----+
```

- 在使用 RDD 时，必须有 Map/Reduce 的概念，需要高级的程序设计能力，而且没有 Schema，只能指定数据的位置，使用起来较不方便。

```
In [60]: userRDD.map(lambda x: (x[2],1)) \
                .reduceByKey(lambda x,y: x+y).collect()
Out[60]: [(u'M', 670), (u'F', 273)]
```

➤ **DataFrame 与 Spark SQL 比 RDD 更快速**

DataFrame 与 Spark SQL 通过 Catalyst 进行最优化，可以大幅提高执行效率。Python 的语言特性使其使用 RDD 时执行速度比 Scala 慢。但是 Spark Python 使用 DataFrames 时，执行性能几乎与 Spark Scala 使用 DataFrames 相同。而且使用 DataFrames 时，无论是 Python 还是 Scala，运行时间都明显比使用 RDD 少很多。

1.4 使用 Python 开发 Spark 机器学习与大数据应用

目前 Spark 支持多种语言：Scala、Python、Java、R。开发者可以根据应用的环境来决定使用哪种语言开发Spark 程序，能更弹性地符合开发时的需求。

Spark 支持的语言比较如下：

- Scala 语言是 Spark 的开发语言，所以与 Spark 兼容性最好，执行效率也最高；但是数据分析师使用 Scala 的比较少，学习门槛也比 Python 高。

- Java 是很常用的程序设计语言，运用很广泛；但是使用 Java 开发 Spark 程序，相同的功能，写出来的语句比较冗长，而且数据分析师也不太熟悉 Java。
- R 语言是数据分析中很常用的语言，数据分析师也都很熟悉，也很容易使用，但是目前 Spark 的 R 语言功能比较不完整，未来 Spark 仍会持续开发以便增加 R 语言的支持。然而 R 主要是数据分析语言，不是通用的程序设计语言，如果还需要其他功能，例如网站整合、网络爬虫等，就需要使用其他的语言。
- Python 是数据分析很常用的程序设计语言，程序代码简明、易学习、高生产力，同时还是面向对象、函数式的动态语言，应用非常广泛。再加上数据分析的相关模块，例如 NumPy、Matplotlib、Pandas、Scikit-learn，让 Python 成为数据分析的主要语言之一。Python 是通用性的程序设计语言，可以用于开发大数据相关的网站、网络爬虫等。

本书主要介绍如何使用 Python 开发 Spark 机器学习与大数据应用。

1.5 Python Spark 机器学习

Python 机器学习模块主要是 Pandas、Scikit-learn，但是在大数据时代有大量的数据，必须具有分布式存储以及分布式计算才能够处理。

有了 Spark 之后，使用 Python 开发 Spark 应用程序，可以使用 HDFS 分布式存储大量数据。还可以使用在多台计算机所建立的集群（例如：Spark stand alone、Hadoop YARN、Mesos）上来执行分布式计算。再加上 Spark 特有的内存运算，让执行速度大幅提升。

如图 1-6 和图 1-7 所示，Spark 机器学习主要有两个 API，说明如下。

➢ Spark MLlib: RDD-based **机器学习** API

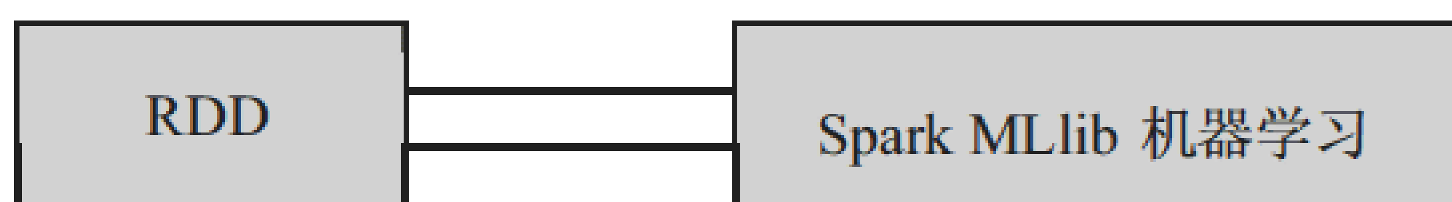


图 1-6　Spark MLlib

Spark 在一开始就提供了以 RDD 为基础的机器学习模块，优点是可以发挥 in-memory 与分布式运算，大幅提升需要迭代的机器学习模块的执行效率，功能强大，能完成 Spark 所有功能。本书第 12~18 章将介绍 Spark MLlib: RDD-based 机器学习。

➢ Spark ML pipeline: Dataframes-based **机器学习** API

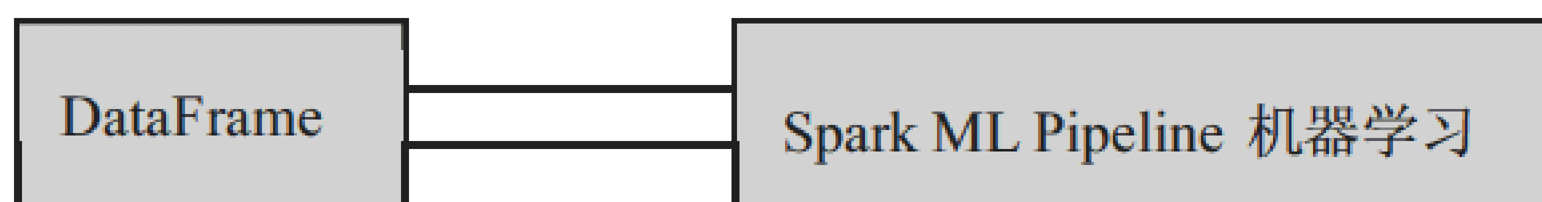


图 1-7　Spark ML Pipeline

DataFrame 与 Spark ML Pipeline 机器学习 API 的设计由来如下：

- DataFrame　Spark 受 Pandas 程序包启发所设计的数据处理架构。
- Spark ML Pipeline　Spark 受 Scikit-learn 程序包启发所设计的机器学习架构。

Spark 受 Pandas 与 Scikit-learn 启发所设计的 DataFrame 与 Spark ML Pipeline 的优点如下：

- Pandas 与 Scikit-learn 都是 Python 很受欢迎的数据分析程序包，很多数据科学家与分析师都很熟悉，采用类似的架构，可降低学习门槛，对于 Spark 的推广有很大的帮助。
- Spark DataFrame 与 Pandas DataFrame 是可以互相转换的，为 Python 开发者带来很大的方便。例如，我们可以把数据读取到 Spark DataFrame，然后使用 Spark ML Pipeline 机器学习、训练、预测，再将结果存回 Spark DataFrame，最后转换为 Pandas DataFrame。转换后就可以运用 Python 丰富的数据可视化程序包（例如 matplotlib、Bokeh 等）进行数据的可视化设计了。
- Spark DataFrame 提供的 API 可轻松读取大数据中的各种数据源，例如 Hadoop、Parquet、JSON 等，还可以通过 JDBC 读取关系数据库，例如 MySQL、MSSQL 等。

Spark DataFrame ML Pipeline 与 Python 常用数据分析软件包如图 1-8 所示。

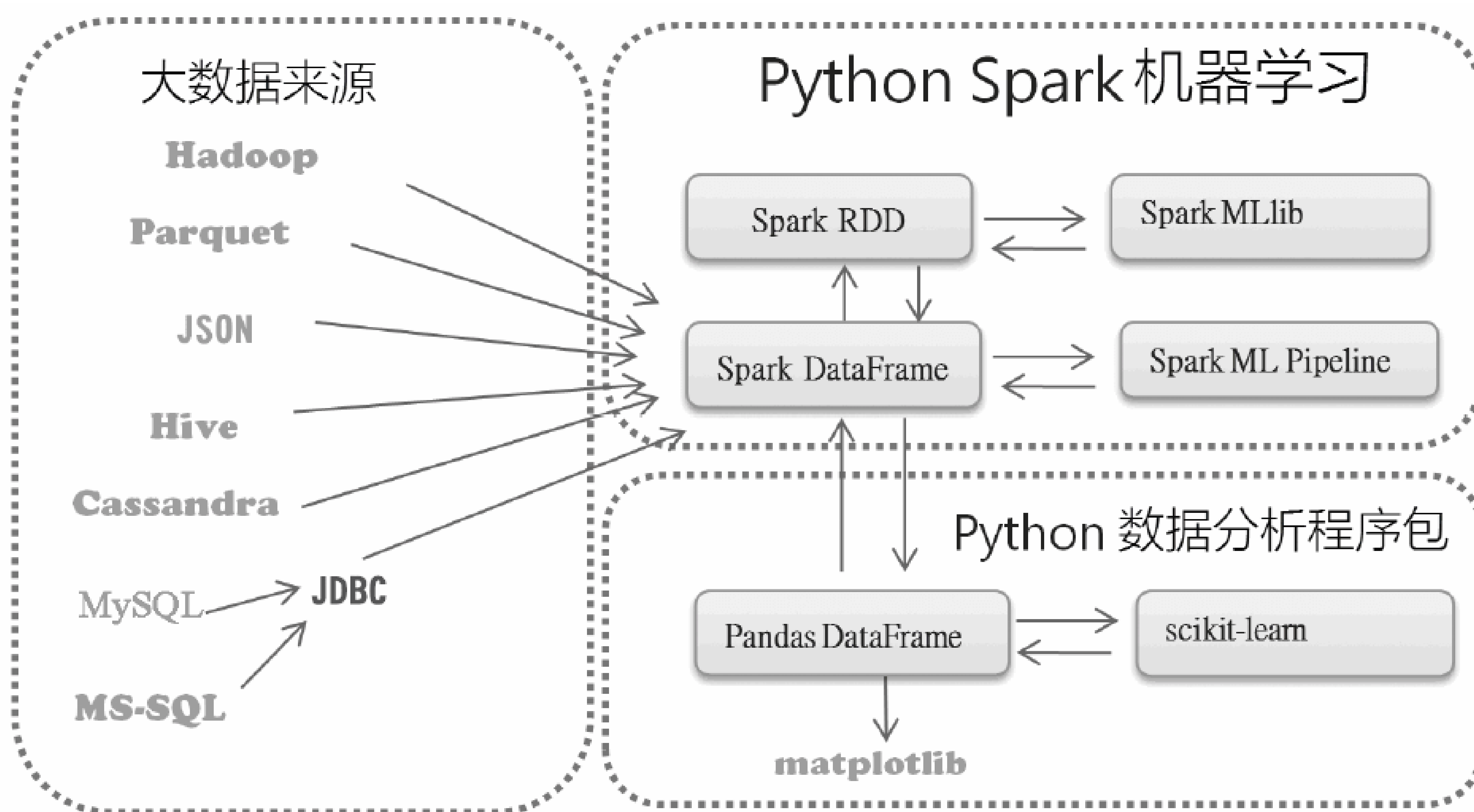


图 1-8　常用数据分析软件包

第 19~22 章将介绍 DataFrame 与 Spark ML Pipeline 机器学习。

1.6　Spark ML Pipeline 机器学习流程介绍

Pipeline 一词的原意是管道、输油管道的意思。

➢ 输油管道 Pipeline

想象一下我们印象中石油化工厂的输油管道，一连串的处理步骤，以管道连接起来。石油从原油开始在管道中流动处理，最后流出的产品是汽油，如图 1-9 所示。

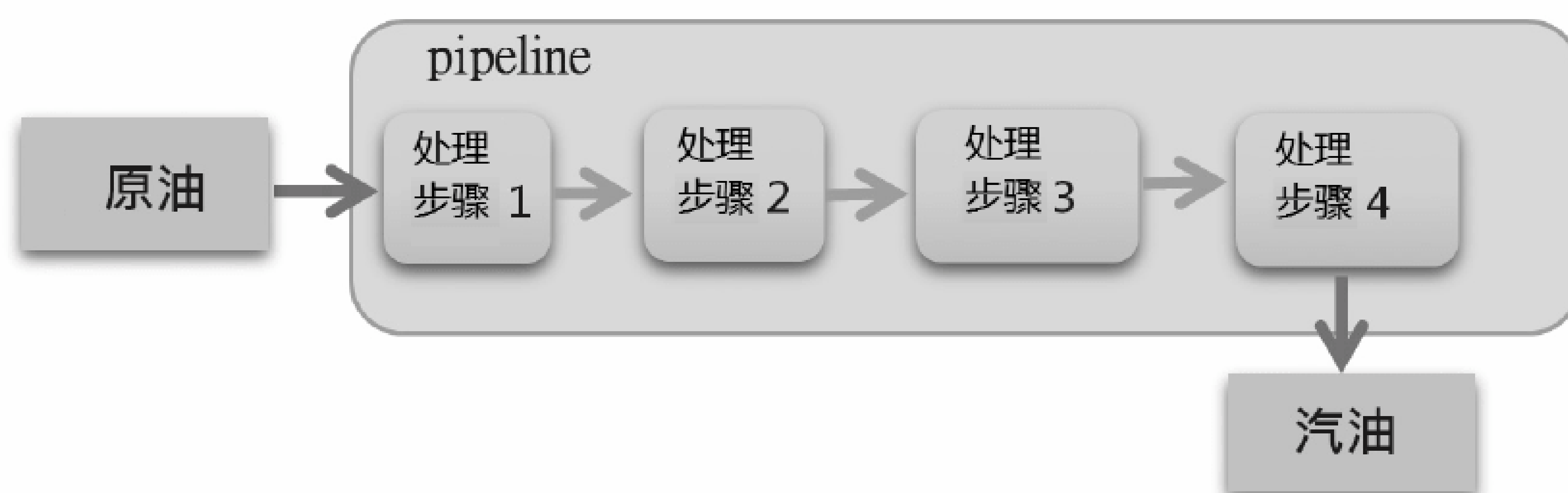


图 1-9　石油生产流程

➢ Spark ML Pipeline 机器学习工作流程

Spark ML Pipeline 机器学习工作流程的原理与石油管道类似，就是将机器学习的每一个阶段建立成 Pipeline 流程：原始数据像数据流一样，经过一连串的处理，例如数据处理、进行训练、建立模型、进行预测，最后产生预测结果。Spark ML Pipeline 的流程如图 1-10 所示。

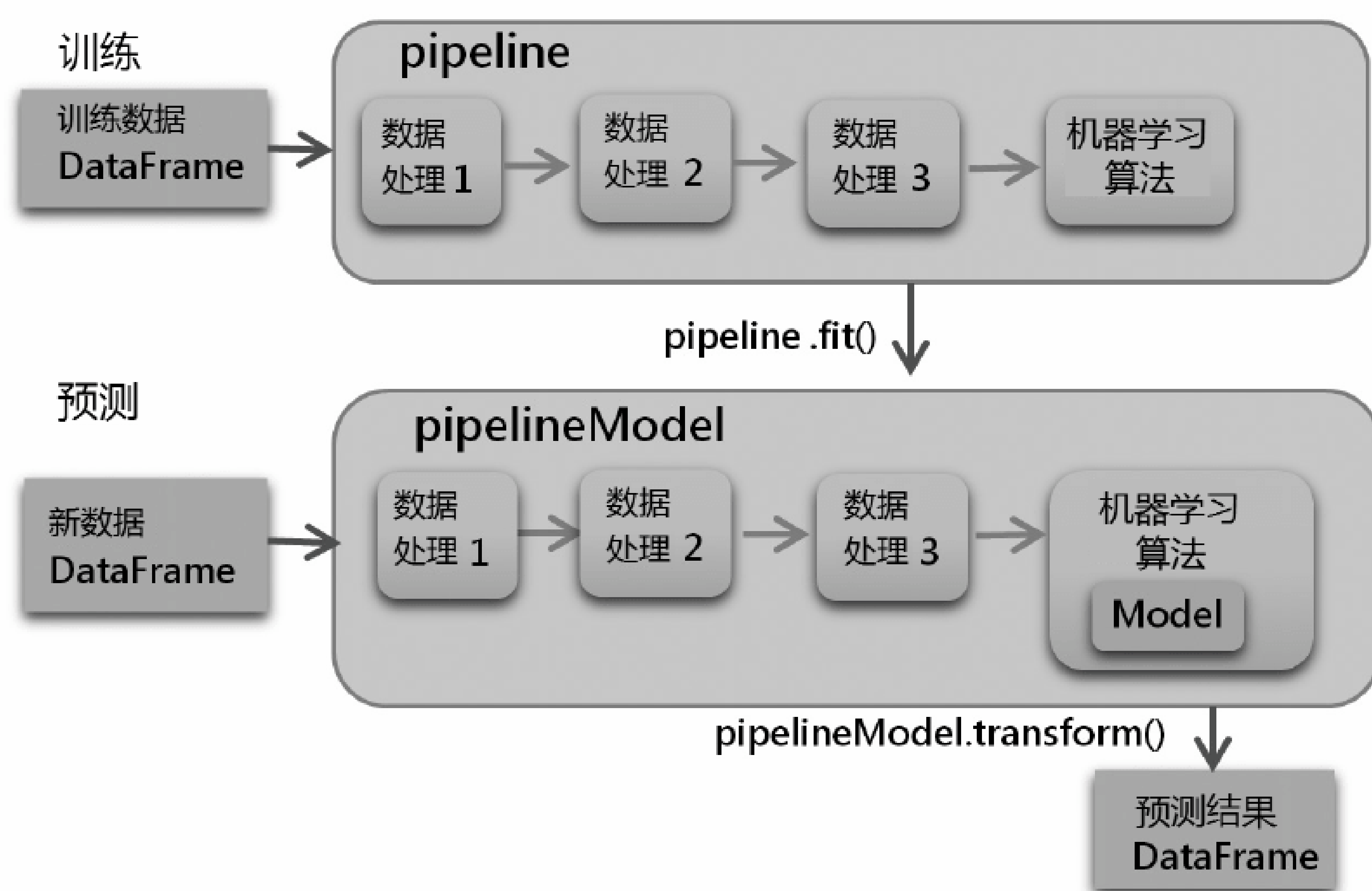


图 1-10　Spark ML Pipeline 机器学习流程

对图 1-10 的说明如下：

（1）**建立机器学习流程 pipeline：** pipeline 可以包含多个阶段（stages），例如在图 1-10 中前 3 阶段是数据处理、第 4 阶段是机器学习算法。

（2）**训练：** “训练数据 DataFrame”使用 pipeline .fit() 进行训练。系统会按照顺序执行每一个阶段，最后产生 pipelineModel 模型。pipelineModel 与 pipeline 类似，只是多了训练后建立的模型 Model。

（3）**预测：**“新数据 DataFrame”使用 pipelineModel.transform() 进行预测。系统会按照顺序执行每一个阶段，并使用机器学习算法模型（Model）进行预测。预测完成后，会产生“预测结果 DataFrame”。

➢ **Spark ML Pipeline 机器学习工作流程的好处**

使用 Spark ML Pipeline 机器学习工作流程的好处是减轻数据分析师程序设计的负担：

- **DataFrame 数据格式一致：**原始数据、数据处理过程、预测结果的数据格式都是 DataFrame。
- **数据处理模块化：**因为有很多数据处理阶段，例如分类特征字段的处理，都已经模块化，只需要套用即可。
- **机器学习算法置换：**已经内建很多机器学习算法，相同的数据处理流程，很容易置换成不同的机器学习算法。

1.7 Spark 2.0 的介绍

Spark 2.0 的新功能介绍如下：

➢ **提升执行性能**

（1）Spark SQL 在 Spark 2.0 可以执行所有 99 TPC-DS 查询，能够执行 SQL:2003 标准的新功能，支持子查询。

（2）SparkSession：新增 SparkSession，同时具备了 SQLContext 与 HiveContext 功能。不过为了向后兼容，我们仍可以使用 SQLContext 与 HiveContext。

➢ **Spark ML pipeline 机器学习程序包**

（1）以 DataFrame 为基础的机器学习程序包 Spark ML pipeline 将成为主要的机器学习架构。过去 Spark MLlib 程序包仍然可以继续使用，未来的开发会以 Spark ML pipeline 为主。

（2）Spark ML pipeline 程序包可以存储、加载训练完成的模型。

➢ **DataSet API**

过去的 DataFrames API 只能执行具有类型的方法（例如 select、groupBy），而 RDD 只能执行不具有类型的方法（例如 map、filter、groupByKey）。有了 Datasets API 之后，就可以同时执行具有类型的方法与不具有类型的方法。不过因为编译时类型安全（compile- time type-safety）并不是 Python 和 R 的语言特性，所以 Python 和 R 不提供 Datasets API。

➢ **Structured Streaming APIs**

Spark 2.0 的 Structured Streaming APIs 是新的结构化数据流处理方式。它可以使用 DataFrame/Dataset API，以 Catalyst 优化提升性能，并且整合了 streaming 数据流处理、

interactive 互动查询与 batch queries 批次查询。

➢ **其他新增功能**

（1)R 语言的分布式算法，增加了 Generalized Linear Models（GLM）、Naive Bayes、Survival Regression 与 K-Means 等算法。

（2）更简单、更高性能的 Accumulator API，拥有更简洁的类型结构，而且支持基本类型。

本书将详细介绍 Spark 2.0 的安装，并且所有范例程序都能在 Spark 2.0 上运行，同时会特别介绍 DataFrame 与 Spark ML Pipeline 机器学习 API 等新功能。

1.8 大数据定义

大数据（Big data）又称为巨量资料、巨量数据或海量数据。一般来说，大数据的特性可归类为 3V：Volume、Variety 和 Velocity。

➢ **Volume（大量数据）**

- **累积庞大的数据**：因特网、企业 IT、物联网、社区、短信、电话、网络搜索、在线交易等，随时都在快速累积庞大的数据。
- **数据量等级**：数据量很容易达到 TB（Terabyte，1024 GB）甚至 PB（Petabyte，1024TB）或 EB（Exabyte，1024 PB）的等级。

➢ **Variety（多样性）**

大数据的数据类型非常多样化，可分为非结构化信息和结构化信息。

- **非结构化信息**：文字、图片、图像、视频、音乐、地理位置信息、个人化信息——如社区、交友数据等。
- **结构化信息**：数据库、数据仓库等。

➢ **Velocity（时效性）**

- **数据的传输流动**：随着带宽越来越大、设备越来越多，每秒产生的数据流越来越大。
- **组织必须能实时处理大量的信息**：时间太久就会失去数据的价值，所以数据必须能在最短时间内得出分析结果。

大数据的影响已经深入到各个领域和行业，在商业、经济及其他领域中，将大量数据进行分析后就可得出许多数据的关联性，可用于预测商业趋势、营销研究、金融财务、疾病研究、打击犯罪等。决策行为将基于数据和分析的结果，而不是依靠经验和直觉。

1.9 Hadoop 简介

Hadoop 是存储与处理大量数据的平台，是 Apache 软件基金会的开放源码、免费且广泛使用的软件。Hadoop 的名称来自于原作者孩子的黄色小象玩具。这个名字容易拼字与发音，适合作为软件开发代码，黄色小象因此成为 Hadoop 的标志。

➢ **Hadoop 的发展历史**

2002 年 Doug Cutting 与 Mike Cafarelia 开始进行 Nutch 项目。
2003 年 Google 发表 GFS（Google File System）与 MapReduce 论文。
2004 年 Doug Cutting 开始将 DFS 与 MapReduce 加入 Nutch 项目。
2006 年 Doug Cutting 加入 Yahoo 团队，并将 Nutch 改名为 Hadoop。
2008 年 Yahoo 使用 Hadoop 包含了 910 个集群，对 1TB 的数据排序花了 297 秒。

➢ **Hadoop 的特性（见表 1-5）**

表 1-5　Hadoop 的特性

特性	说明
可扩展性（Scalable）	Hadoop 采用分布式计算与存储，当我们扩充容量或运算时，不需要更换整个系统，只需要增加新的数据节点服务器即可
经济性（Economical）	Hadoop 采用分布式计算与存储，不必使用昂贵高端的服务器，只需一般等级的服务器就可架构出高性能、高容量的集群
弹性（Flexible）	传统的关系数据库存储数据时必须有数据表结构 schema（各个数据对象的集合），然而 Hadoop 存储的数据是非结构化（schema-less）的，也就是说可以存储各种形式、不同数据源的数据
可靠性（Reliable）	Hadoop 采用分布式架构，因此即使某一台服务器硬件坏掉，甚至整个机架坏掉，HDFS 仍可正常运行，因为数据还会有另外两个副本

本书第 2～5 章将介绍 Hadoop 单机模式与多台机器 cluster 模式的安装。

1.10 Hadoop HDFS 分布式文件系统

HDFS 采用分布式文件系统（Hadoop Distributed File System），可以由单台服务器扩充到数千台服务器，如图 1-11 所示。

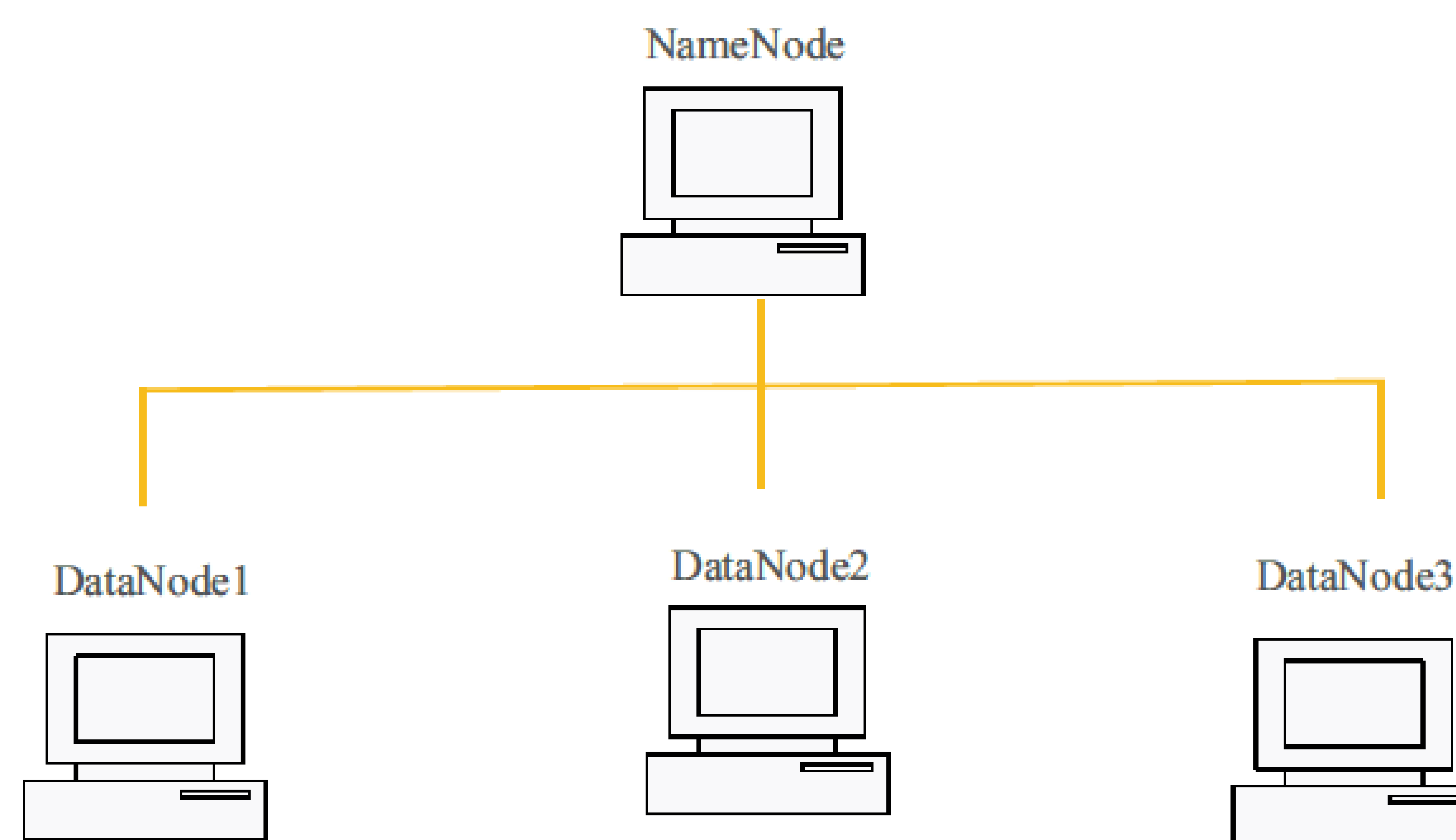


图 1-11　分布式文件系统

- NameNode 服务器负责管理与维护 HDFS 目录系统并控制文件的读写操作。
- 多个 DataNode 服务器负责存储数据，在图 1-11 中我们只列出 3 个 DataNode，实际上大型的集群可以有成千上万个节点。

➢ HDFS 设计的前提与目标

- 硬件故障是常态而不是异常（Hardware Failure）

HDFS 是设计运行在低成本的普通服务器上的。硬件故障是常态，而不是异常，所以 HDFS 被设计成具有高度容错能力、能够实时检测错误并且自动恢复，这是 HDFS 核心的设计目标。

- Streaming 流式数据存取（Streaming Data Access）

运行在 HDFS 上的应用程序会通过 Streaming 存取数据集。HDFS 的主要设计是批处理，而不是实时互动处理，优点是可以提高存取大量数据的能力，但是牺牲了响应时间。

- 大数据集（Large Data Sets）

为了存储大数据集，HDFS 提供了 cluster 集群架构，用于存储大数据文件，集群可扩充至数百个节点。

- 简单一致性模型（Simple Coherency Model）

HDFS 的存取模式是一次写入多次读取（write-once-read-many），一个文件被创建后就不会再修改。这样设计的优点是：可以提高存储大量数据的能力，并简化数据一致性的问题。

- **移动“计算”比移动“数据”成本更低**（Moving Computation is Cheaper than Moving Data）

当我们的 cluster 集群存储了大量的数据时，要搬移数据必须耗费大量的时间成本。因此如果我们有“计算”数据的需求时，就会将“计算功能”在接近数据的服务器中运行，而不是搬移数据。

- 跨硬件与软件平台

HDFS 在设计时就考虑到平台的可移植性，这种特性有利于 Hadoop 的推广。

➢ **HDFS 文件存储架构（见图 1-12）**

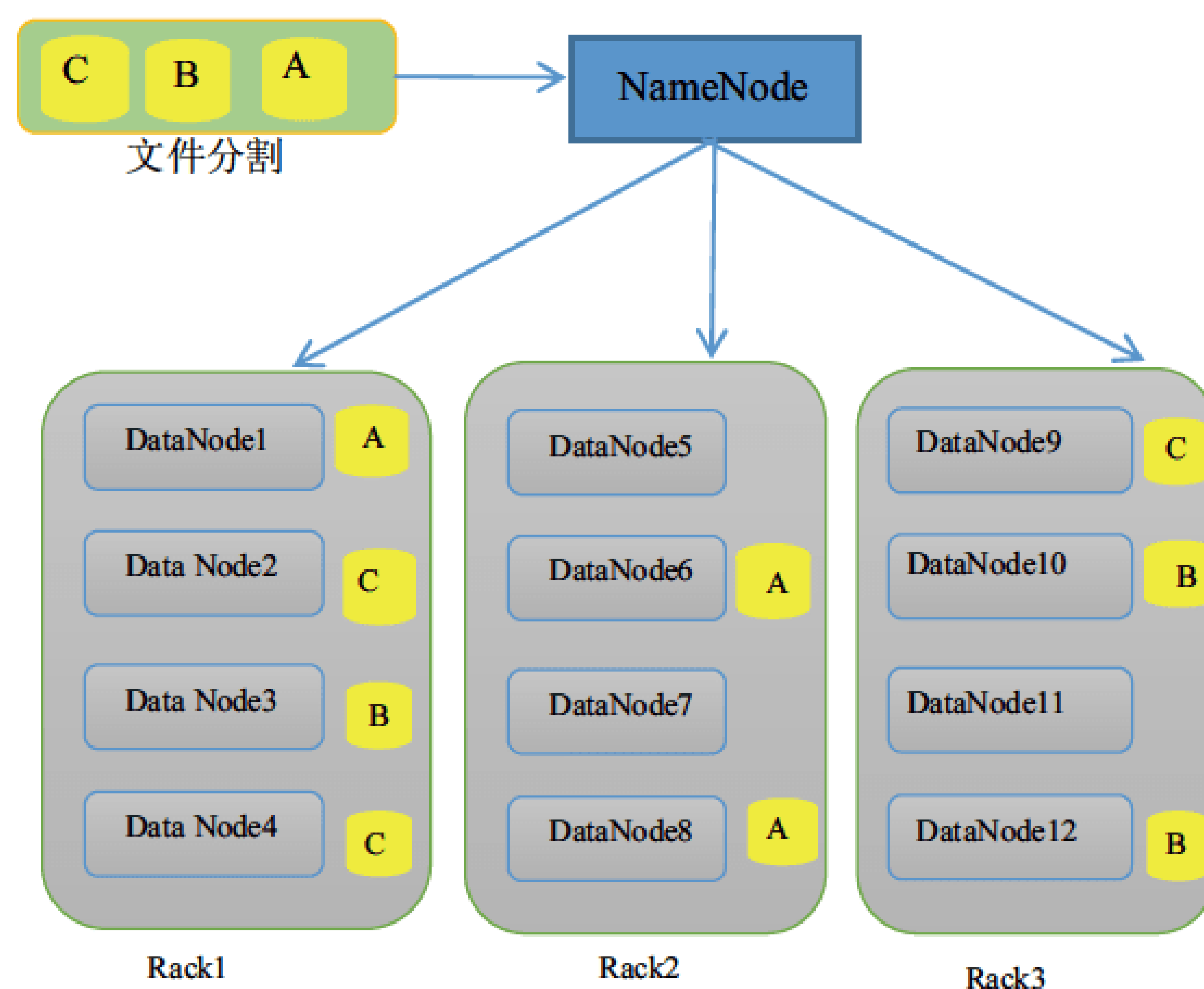


图 1-12　文件存储架构

➢ **文件分割**

当用户以 HDFS 命令要求存储文件时，系统会将文件切割为多个区块（Block），每个区块是 64MB。在图 1-12 中，文件被分割为 A、B、C 共 3 个区块。

➢ **区块副本策略**

- 一个文件区块默认会复制成 3 份，我们可以在 Hadoop 配置中设置文件区块要创建几个副本。
- 文件区块损坏时，NameNode 会自动寻找位于其他 DataNode 上的副本来恢复数据，维持 3 份的副本策略。

➢ **机架感知**

- 在图 1-12 中，共有 Rack1、Rack2、Rack3 共 3 个机架，每个机架都有 4 台 DataNode 服务器。
- HDFS 具备机架感知功能，如图 1-12 所示。以 Block C 为例：第一份副本放在 Rack1 机架的节点，第二份放在同机架 Rack1 的不同节点，最后一份放在不同机架 Rack3 的不同节点。
- 这样设计的好处是防止数据遗失，有任何机架出现故障时仍可以保证恢复数据，提高网络性能。

本书第 6 章将介绍 Hadoop HDFS 命令。

1.11 Hadoop MapReduce 的介绍

利用大数据进行数据分析处理时数据量庞大，所需的运算量也巨大。Hadoop MapReduce 的做法是采用分布式计算的技术：

- Map 将任务分割成更小任务，由每台服务器分别运行，如图 1-13 所示。
- Reduce 将所有服务器的运算结果汇总整理，返回最后的结果。

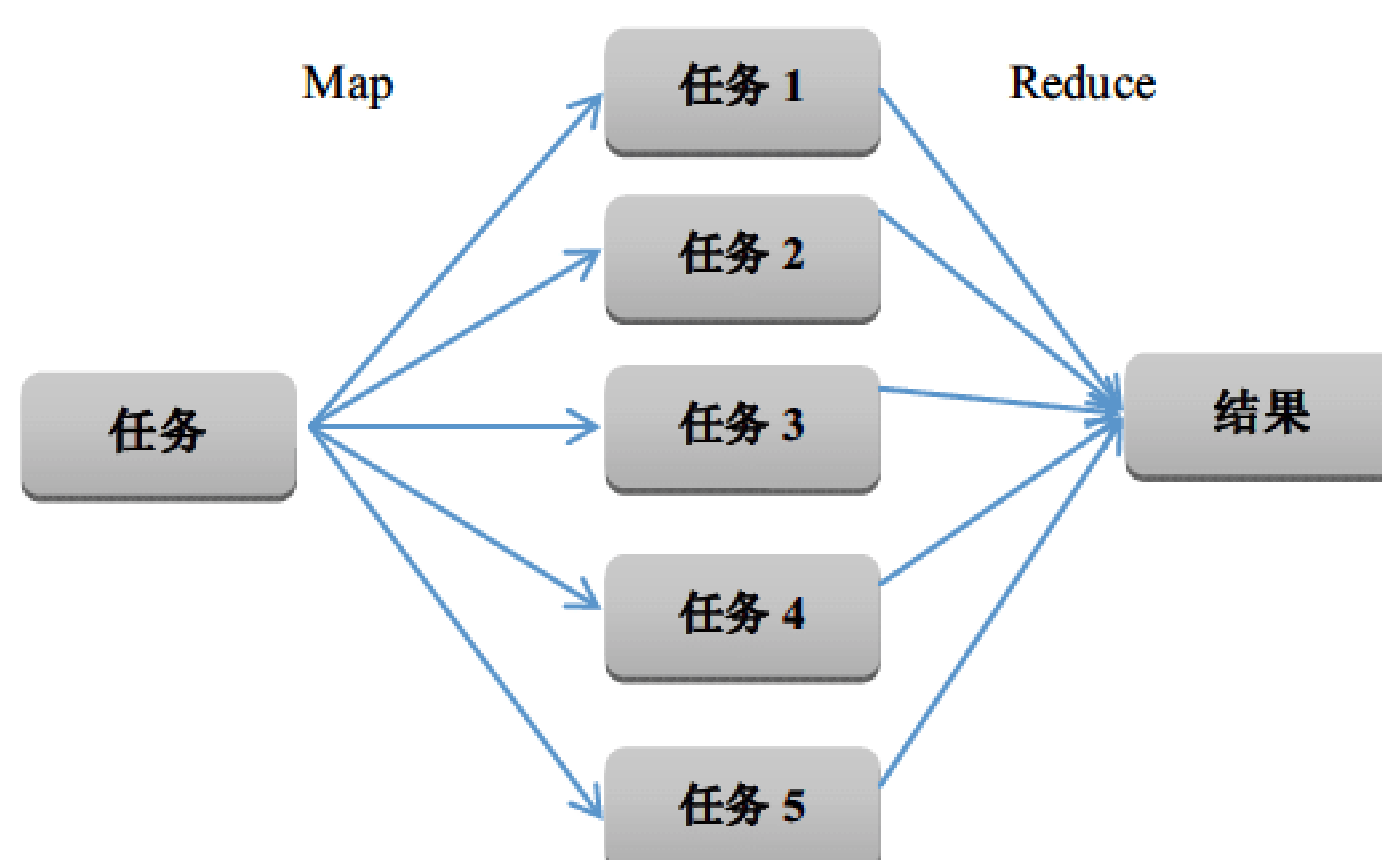


图 1-13　分割任务

通过 MapReduce 方式，可以在上千台机器上并行处理巨量的数据，大大减少数据处理的时间。

➢ MapReduce 版本 2.0 YARN

Hadoop 的 MapReduce 架构称为 YARN（Yet Another Resource Negotiator，另一种资源协调者），是效率更高的资源管理核心。可以到下列网址查看 YARN 架构图（见图 1-14）：http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html。

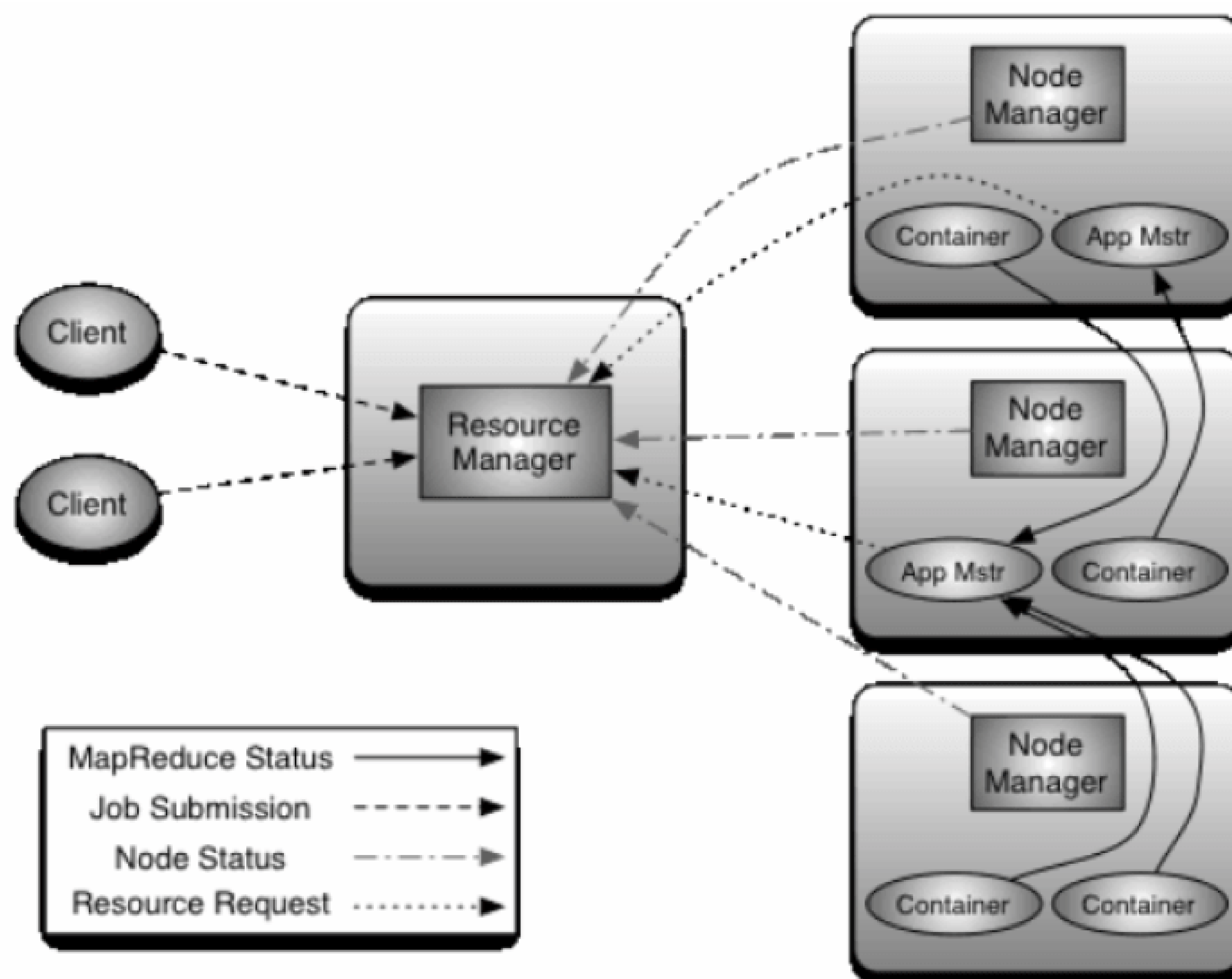


图 1-14　YARN 架构图

从图 1-14 中可以看到：

- 在 Client 客户端，用户会向 Resource Manager 请求执行运算（或执行任务）。
- 在 NameNode 会有 Resource Manager 统筹管理运算的请求。

- 在其他的 DataNode 会有 Node Manager 负责运行，以及监督每一个任务（task），并且向 Resource Manager 汇报状态。

➢ **Hadoop MapReduce 的计算框架**

如图 1-15 所示，Hadoop MapReduce 在运算时需要将中间产生的数据存储在硬盘中。然而，磁盘 I/O 往往是性能的瓶颈，因此会有读写数据延迟的问题。

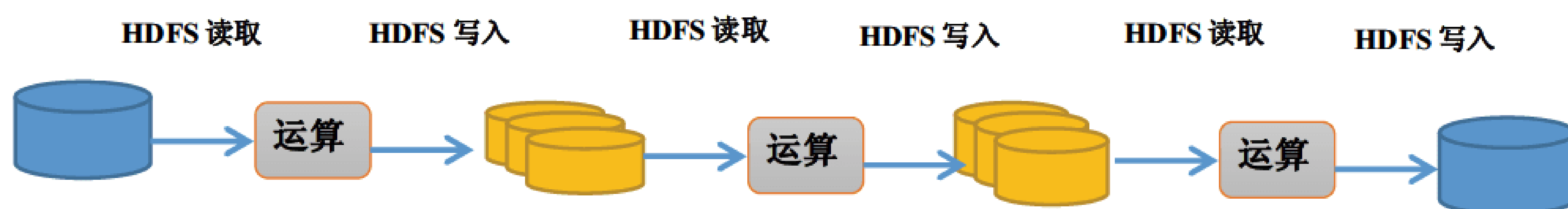


图 1-15　运算过程中需要存储数据

1.2 节所介绍的 Spark 是基于内存内的计算框架，所以不会有磁盘 I/O 读写数据延迟的问题，可以大幅提升性能。

本书第 7 章将介绍 Hadoop MapReduce。

1.12 结论

本章我们介绍了机器学习、Spark 基本概念、Python 开发 Spark 机器学习与大数据应用、Spark ML Pipeline 机器学习流程、大数据定义、Hadoop 基本概念、HDFS、MapReduce 等基本原理，让大家有一个基本概念。接下来我们将在实体计算机上操作，介绍如何安装 Virtual Box 虚拟机。

第 2 章 VirtualBox虚拟机软件的安装

本章将介绍如何安装 Virtual Box 虚拟机软件，让你可以在 Virtual Box 上安装多台 Linux 虚拟机。

本章，我们将介绍 Hadoop 的安装方法。根据 Hadoop 官方说明文件的建议，Hadoop 最主要是在 Linux 操作系统环境下运行。市面上有很多 Open Source 开放源码的操作系统都属于 Linux 操作系统，例如 Fedora、Ubuntu 等。本书将介绍如何安装 Ubuntu，并且在 Ubuntu 上安装 Hadoop 和 Spark。

当我们介绍搭建 Hadoop cluster 集群时，需要多台计算机。但是，大多数的读者使用的是 Windows 操作系统，而且没有很多台计算机。

为了让读者可以上机实践，我们会介绍如何安装 VirtualBox 虚拟机软件，这样就能够新增多台虚拟机，然后在虚拟机中安装 Ubuntu 与 Hadoop 集群了。如果有多台实体计算机，也可以安装在实体计算机中，安装的方法与虚拟机相同。

VirtualBox 是由 Oracle 公司所开发的，是一款免费并且具有中文版的虚拟机（Virtual Machine）软件。VirtualBox 很适合练习操作系统和软件的安装与测试，在虚拟计算机中进行的任何实验都不会影响实体计算机的正常运行。安装 VirtualBox 之后，你可以在计算机新增多台虚拟机，然后在虚拟机中安装不同的操作系统，例如 Windows、Linux 等，使用上与实体计算机一样。

2.1 VirtualBox 的下载和安装

关于 VirtualBox 5.0 的下载、安装以及设置说明如下。

步骤 01 VirtualBox 下载网址

连接到网址（https://www.virtualbox.org/wiki/Downloads），选择“x86/amd64”，下载 VirtualBox Windows 版本，如图 2-1 所示。

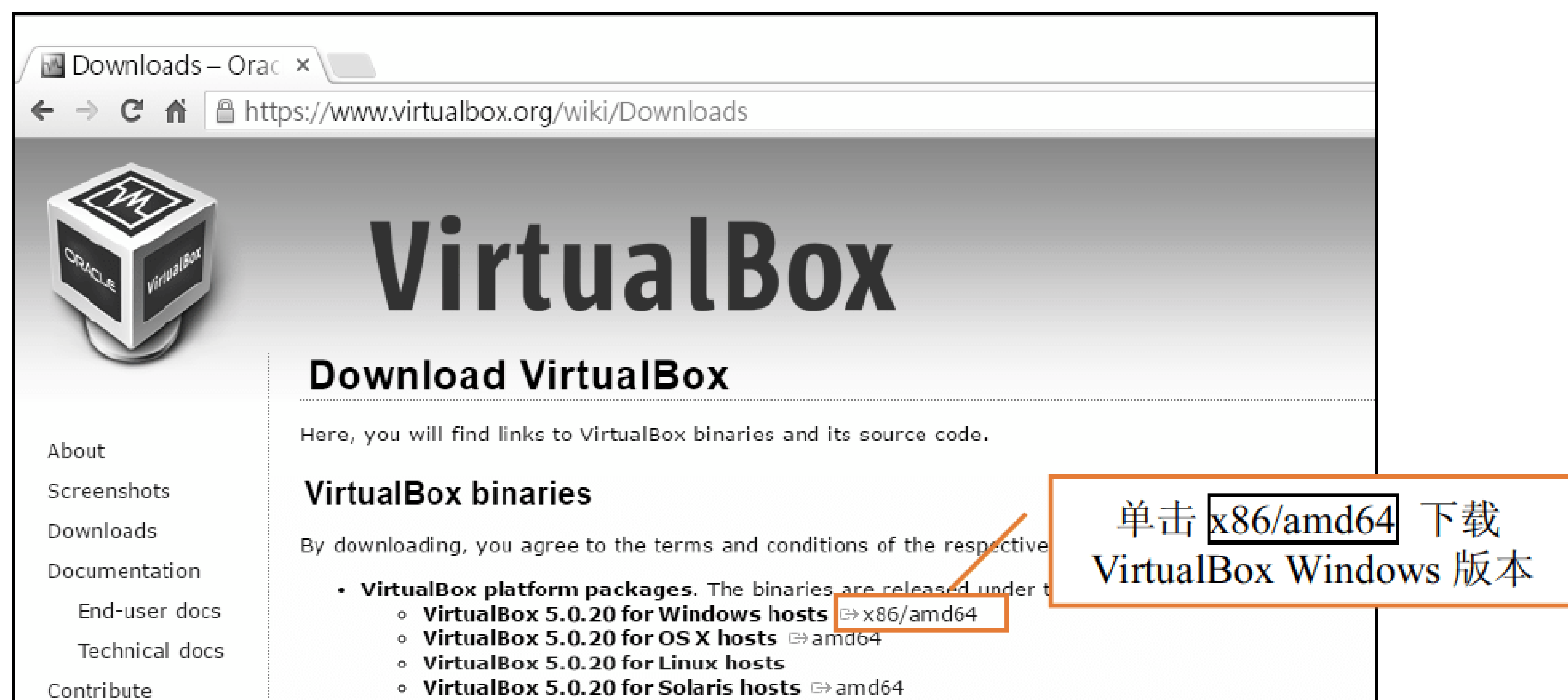


图 2-1　下载 Virtual Box

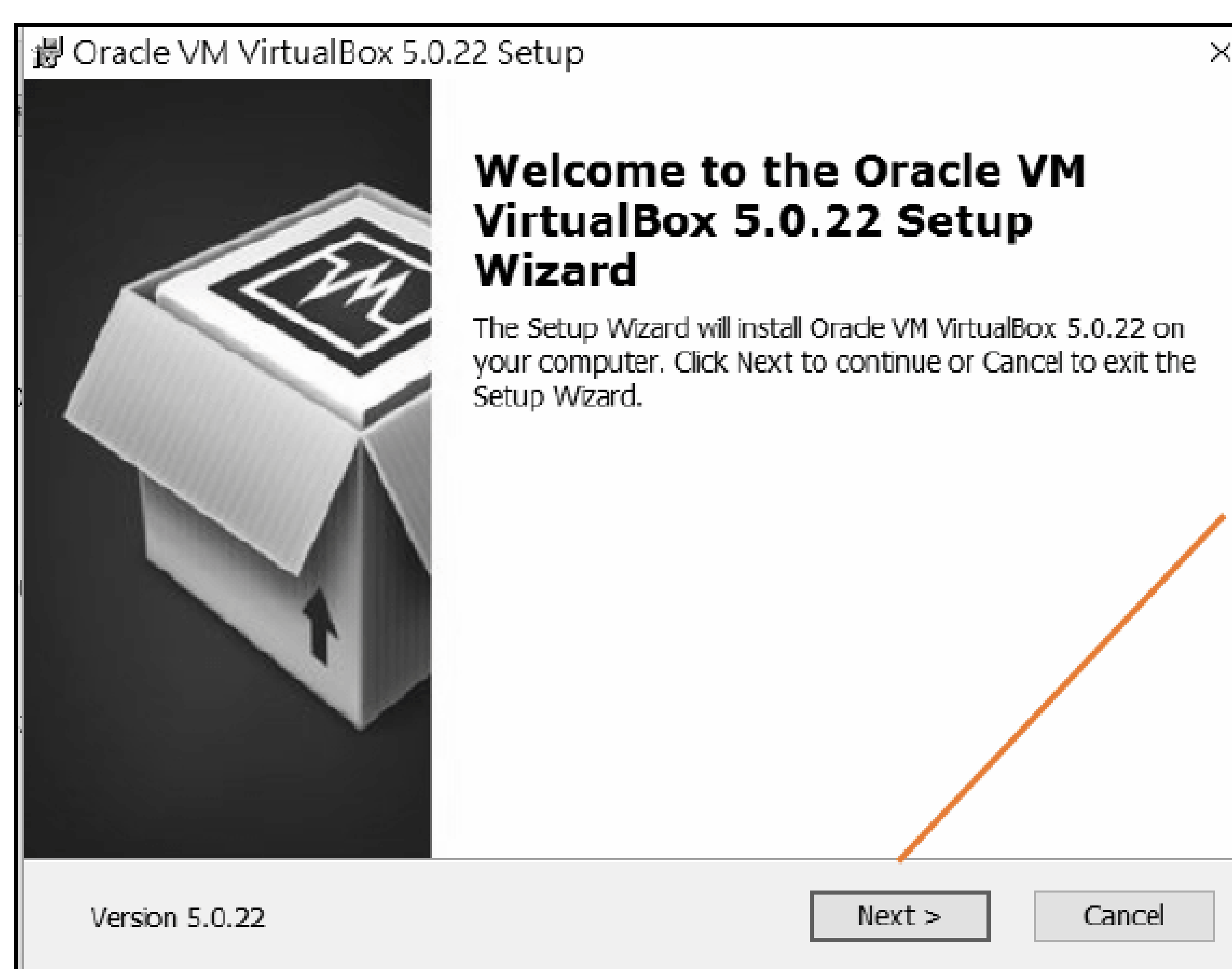
步骤 02 运行 VirtualBox 安装程序（见图 2-2）



下载后在存储的目录中可以看到已下载的安装文件，用鼠标左键双击安装程序的图标，即可开始运行安装程序

图 2-2　开始安装程序

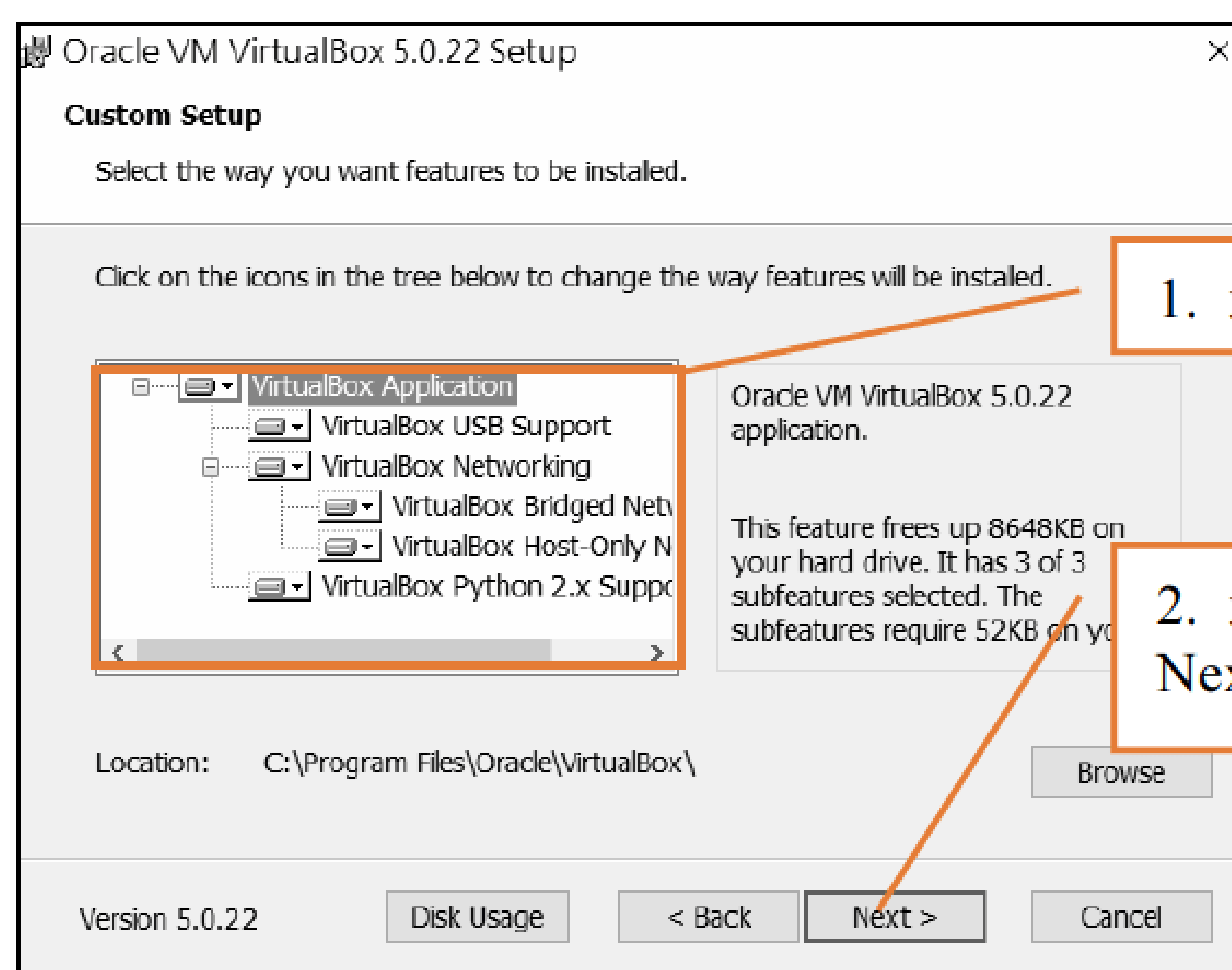
步骤 03　开始安装 VirtualBox（见图 2-3）



单击 Next 按钮开始安装

图 2-3　单击 Next 按钮

步骤 04　选择 VirtualBox 功能（见图 2-4）



1. 选择要安装的功能

2. 选择默认安装，单击 Next 按钮即可

图 2-4　选择安装功能

步骤 05　自定义安装界面（见图 2-5）

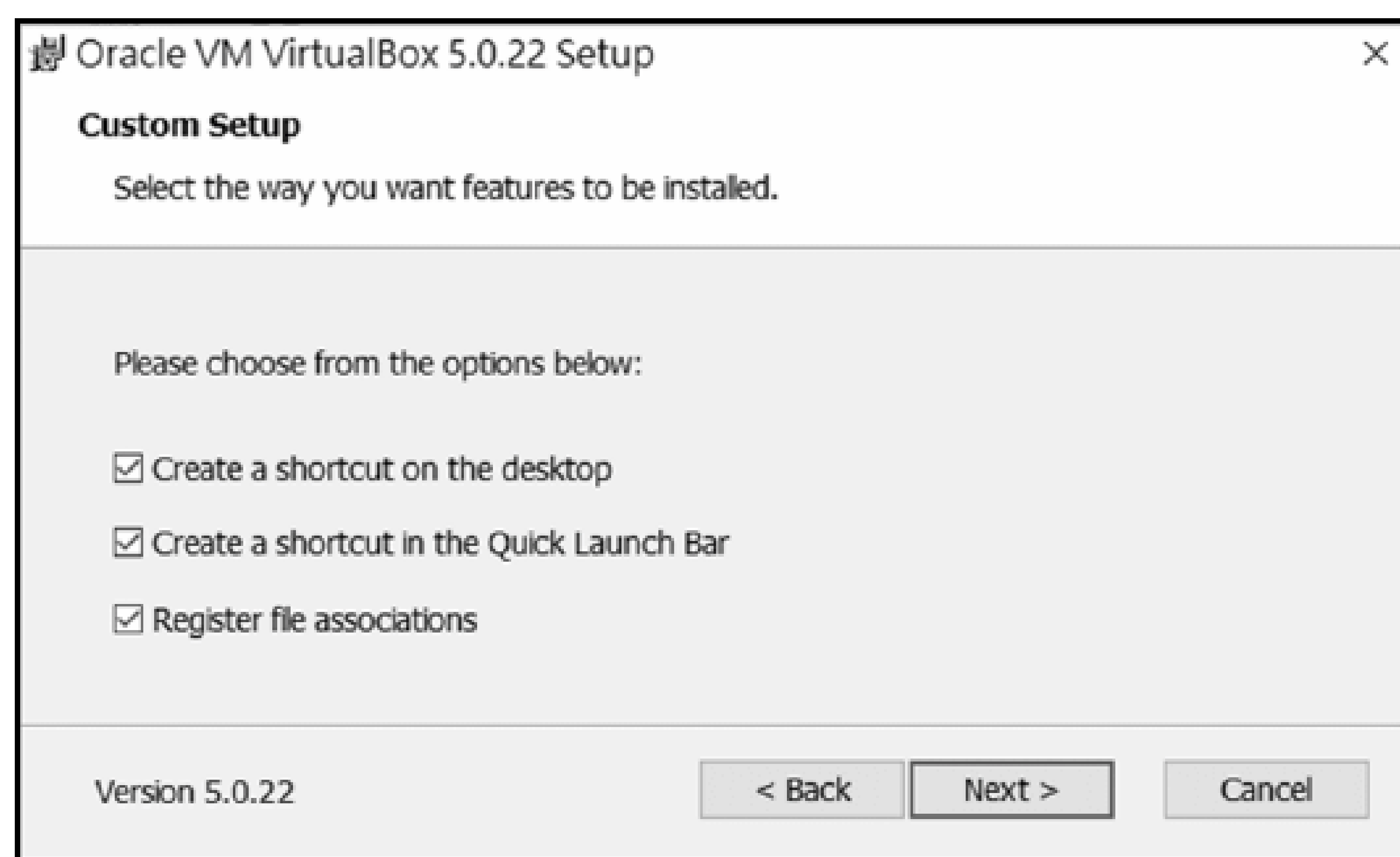


图 2-5　自定义安装

步骤 06　警告界面（见图 2-6）

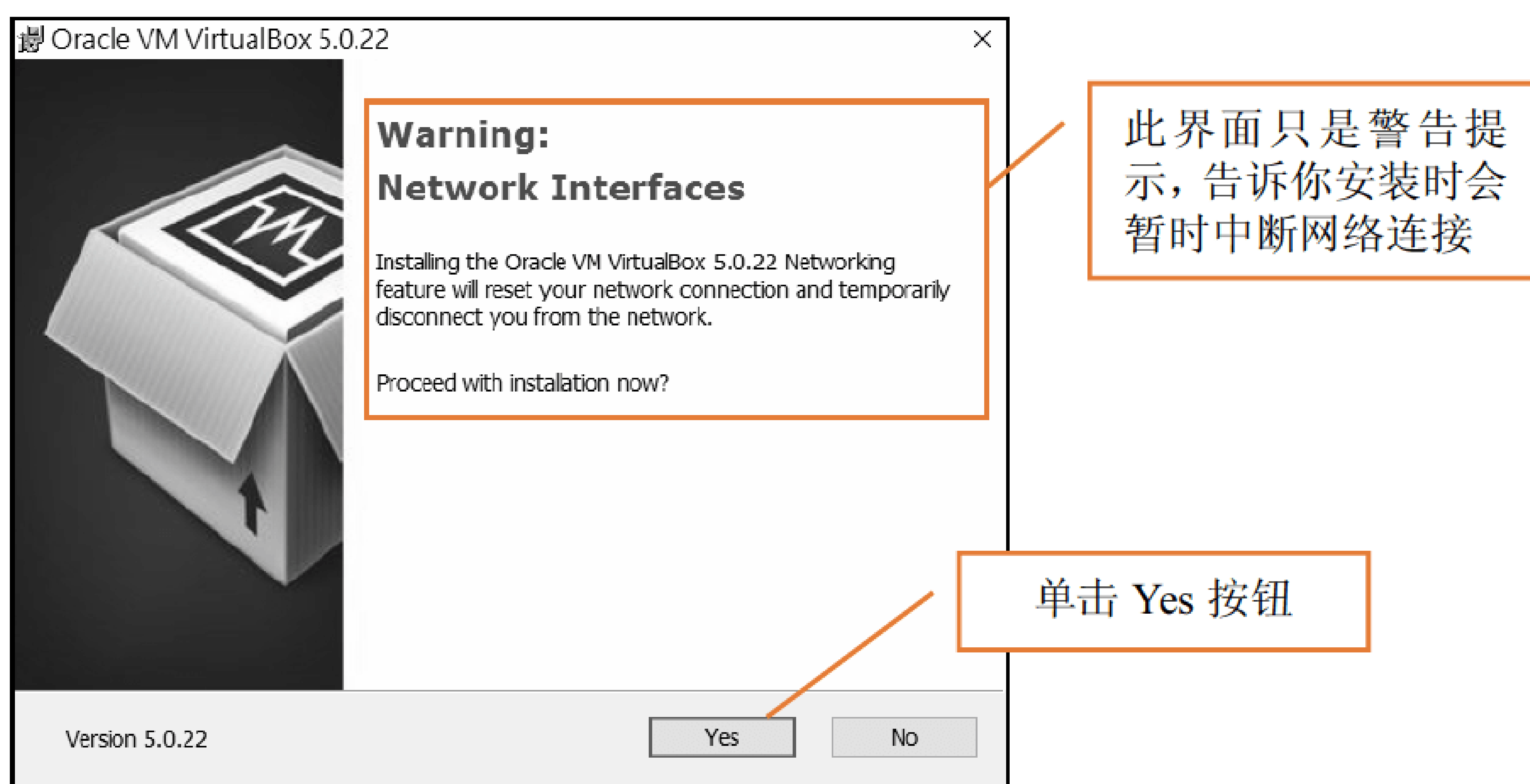


图 2-6　警告界面

步骤 07　完成设置（见图 2-7）

完成设置后，单击 Install 按钮开始安装，也可以单击 Back 按钮回到上一个步骤修改设置。

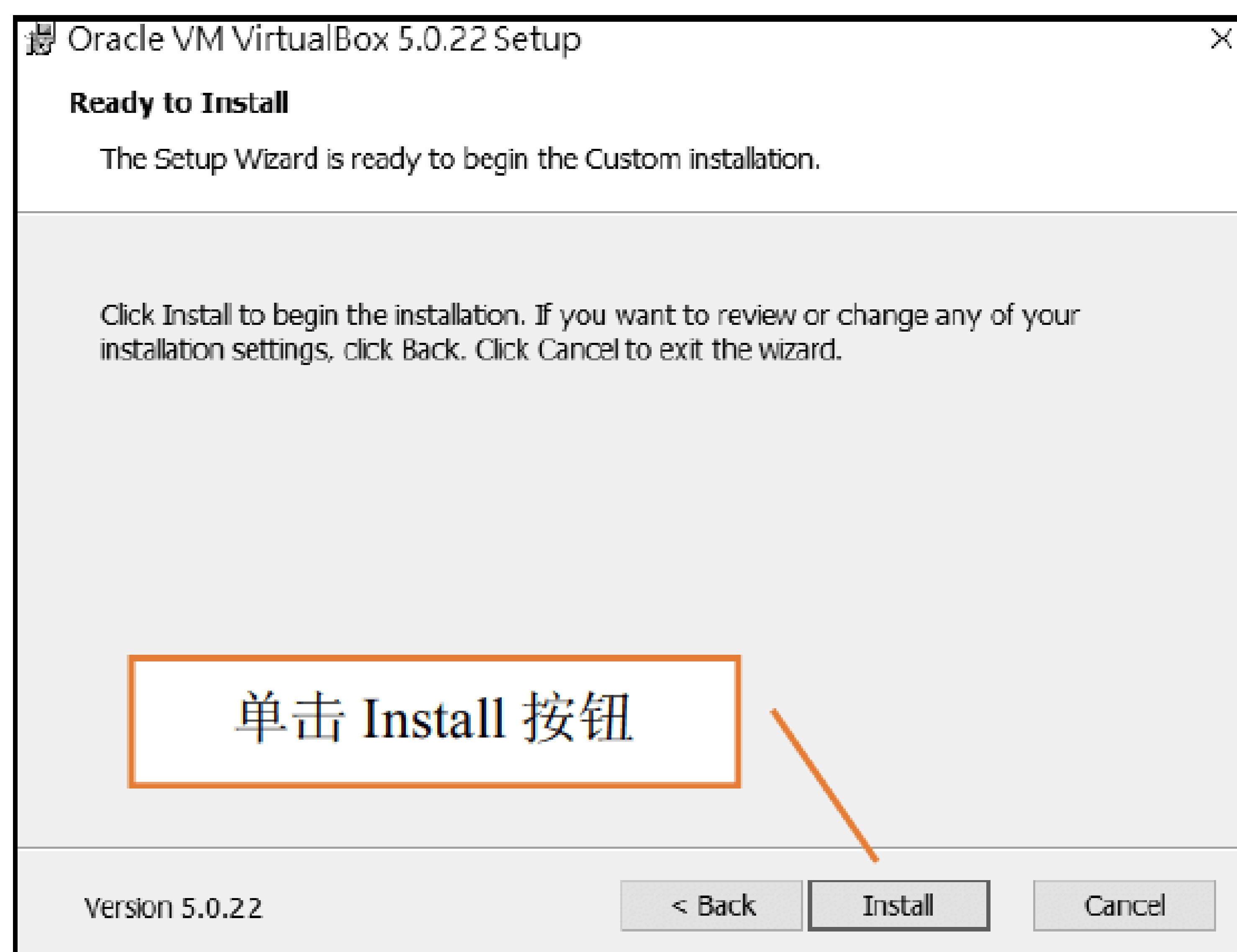


图 2-7　完成设置

步骤 08　安装完成

安装完成后单击 Finish 按钮（见图 2-8），就会启动 VirtualBox。

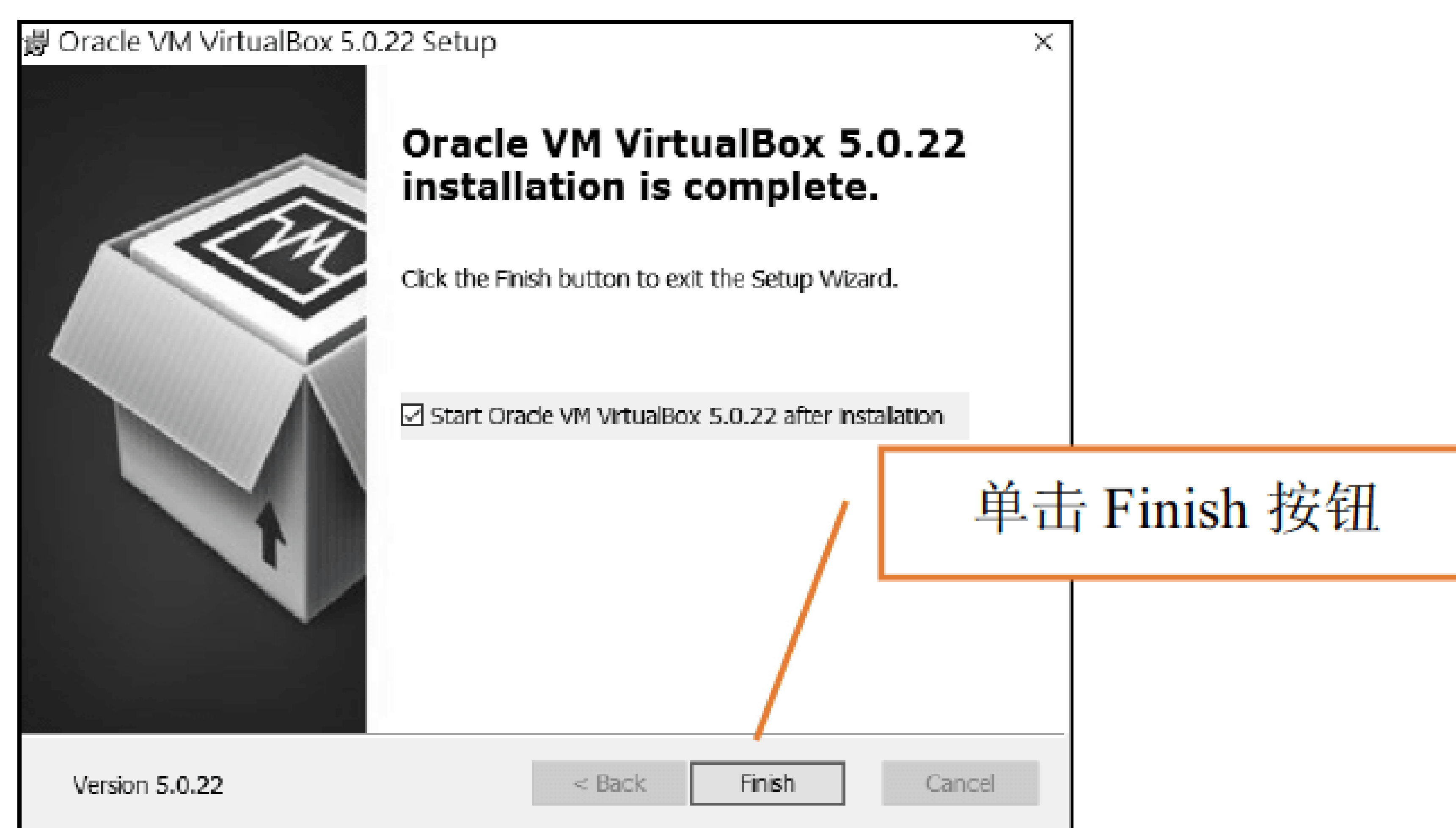


图 2-8　完成安装

步骤 09　启动 VirtualBox 的界面

重新启动后，可以看到 VirtualBox 的屏幕显示界面，如图 2-9 所示。

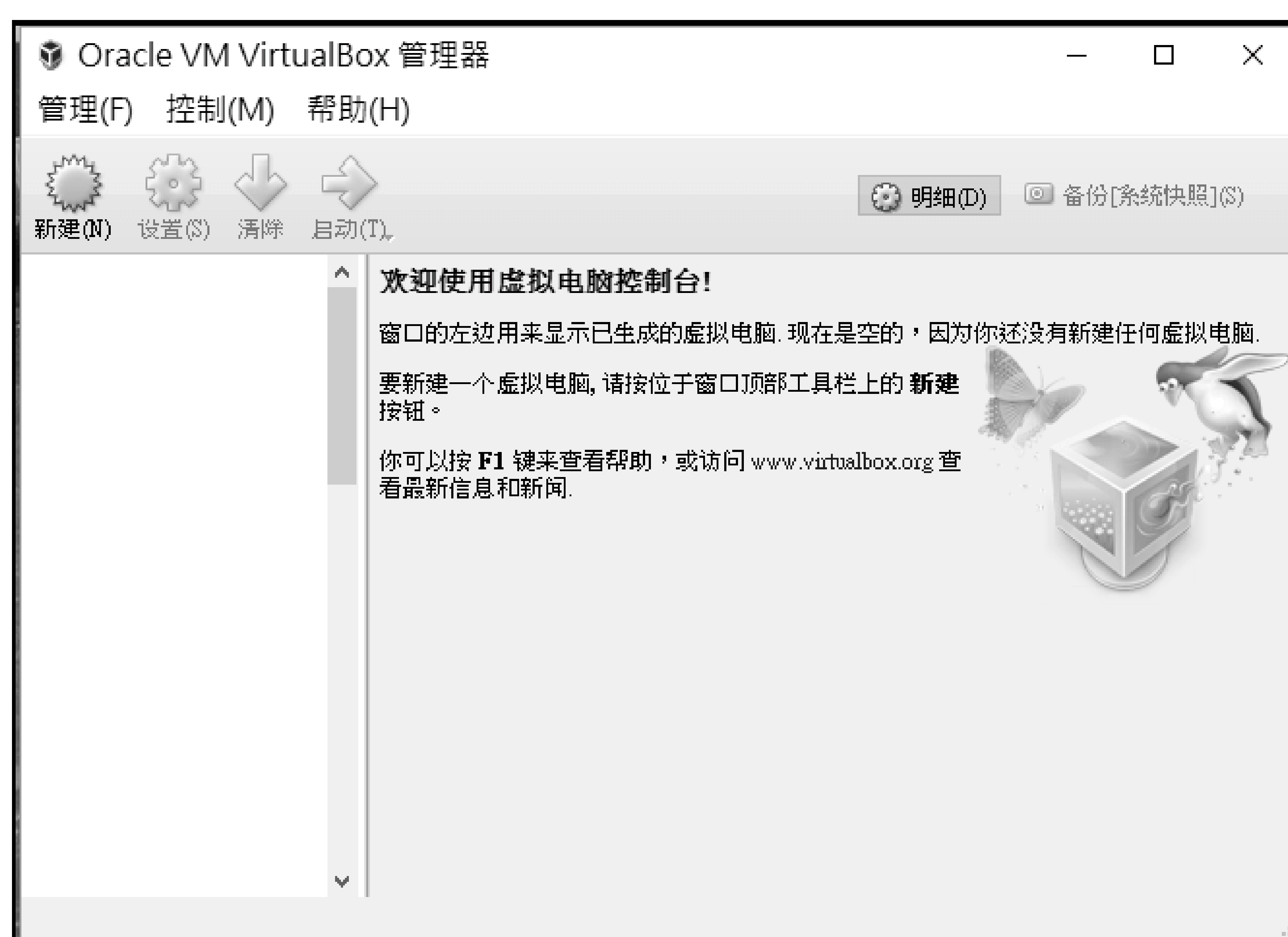


图 2-9　启动界面

2.2　设置 VirtualBox 存储文件夹

当创建虚拟主机时，VirtualBox 会创建一个文件，用于存储这个虚拟主机的所有数据，默认在 C 盘。因为虚拟主机文件通常很占空间，所以建议设置在空闲空间比较大的磁盘上，例如 D 盘或者 E 盘，这样也比较容易进行数据备份。

接下来，我们将介绍如何设置 VirtualBox 默认存储文件夹。

步骤01 首选项设置，选择菜单 File→Preferences...（见图 2-10）

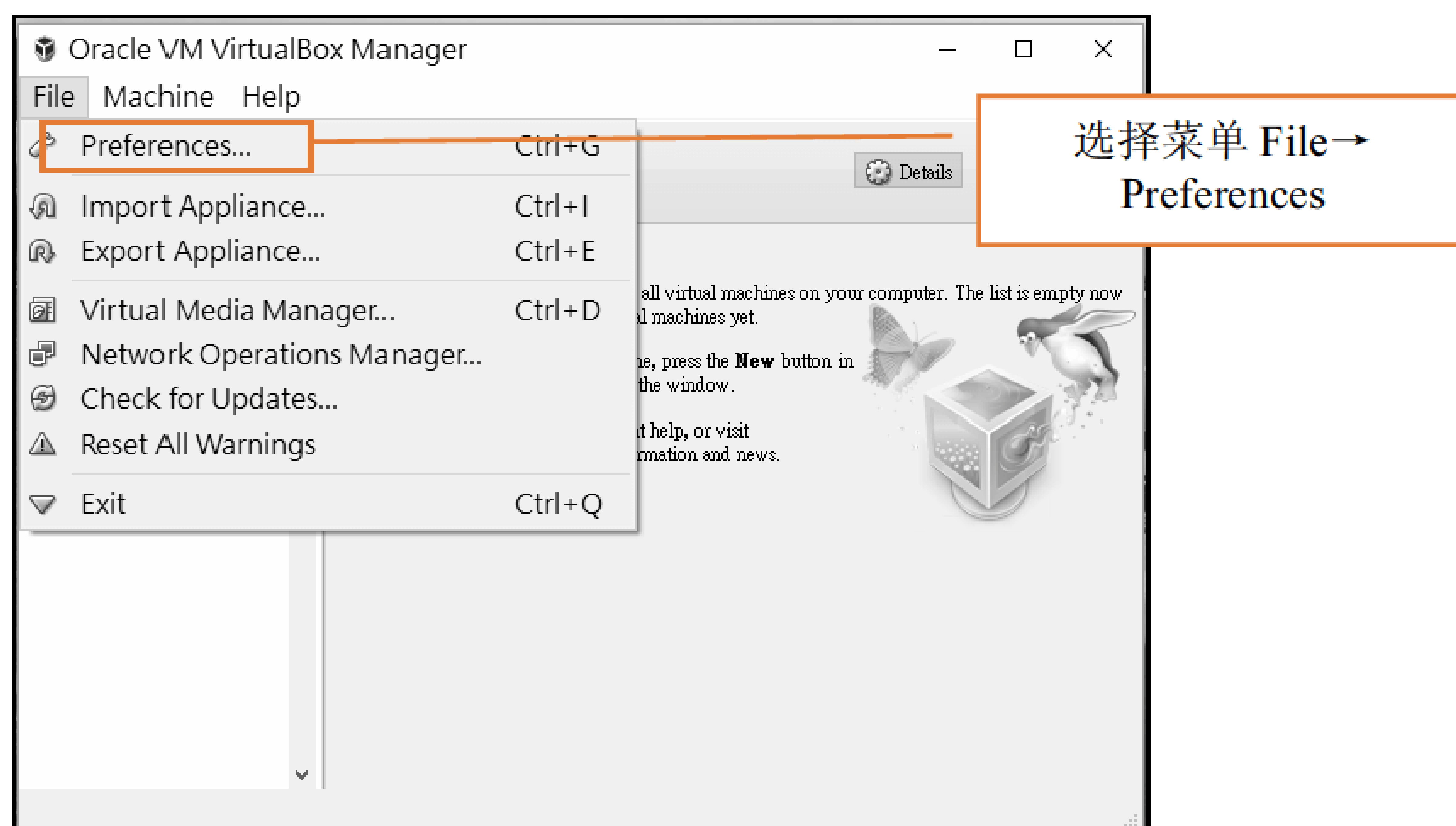


图 2-10　选择菜单项

步骤02 设置界面

选择“常规”设置，在默认机器文件夹的下拉菜单中选择“其它”，如图 2-11 所示。

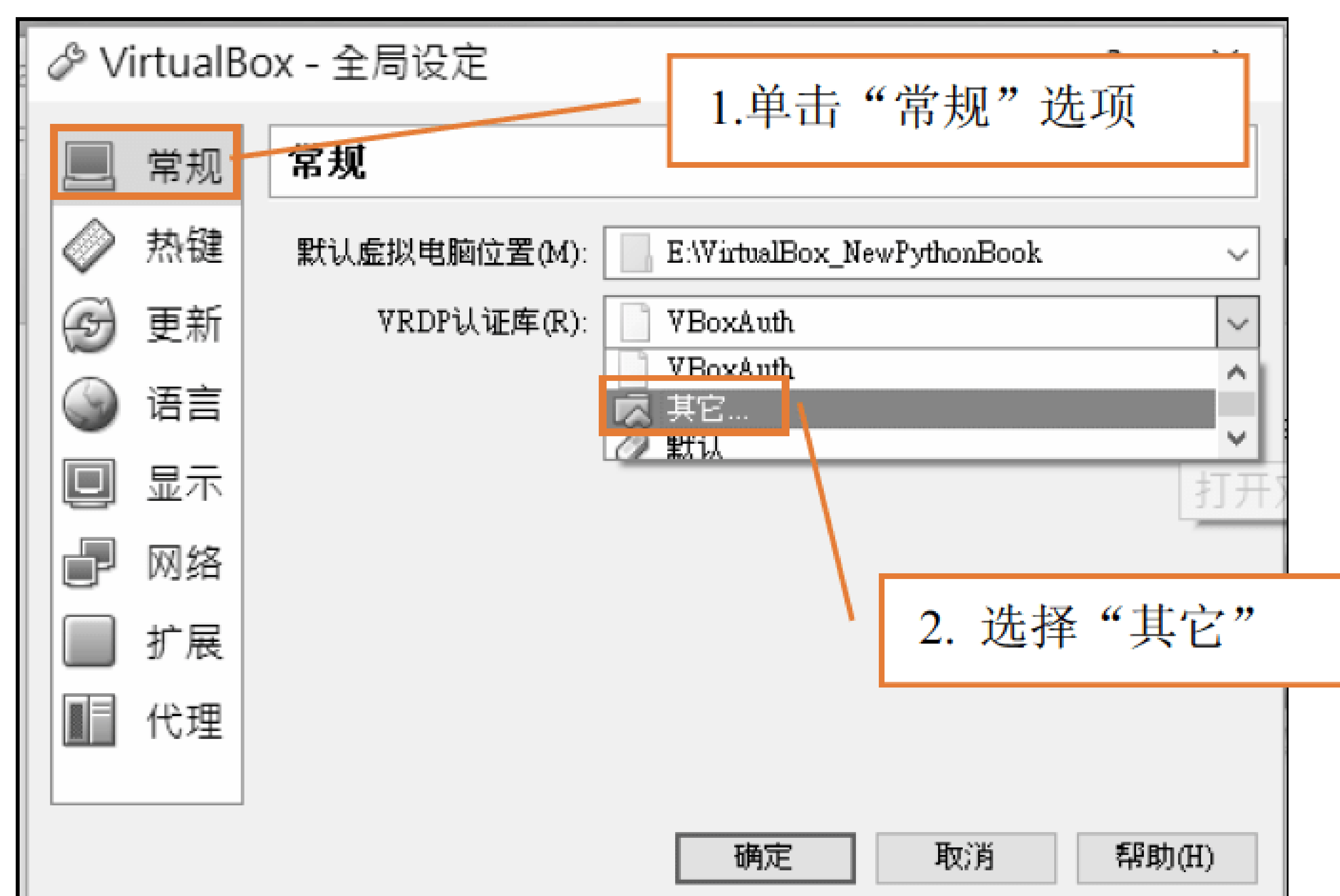


图 2-11　设置选项

步骤03 选择文件夹

选择要存储 VirtualBox 虚拟机文件的文件夹，例如选择我们已经创建好的 E:\VirtualBox，选好后单击“确定”按钮，如图 2-12 所示。

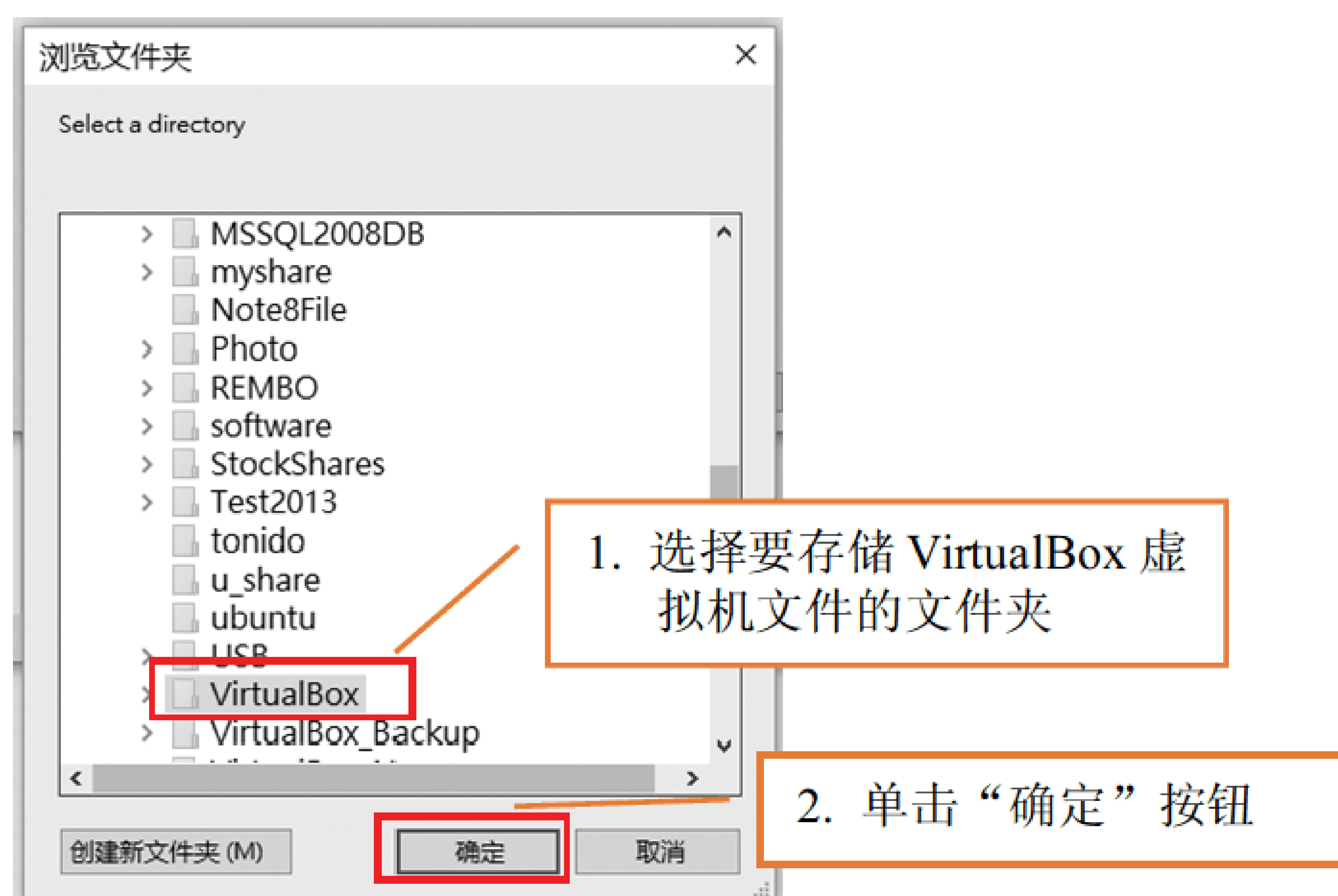


图 2-12　选择文件夹

步骤 04 设置完成（见图 2-13）



图 2-13　设置完成

2.3　在 VirtualBox 创建虚拟机

接下来，我们要新建一台虚拟机。

步骤 01 创建虚拟机

回到 VirtualBox 启动后的界面，单击“新建”按钮，就会出现“新建虚拟电脑”对话框，如图 2-14 所示。

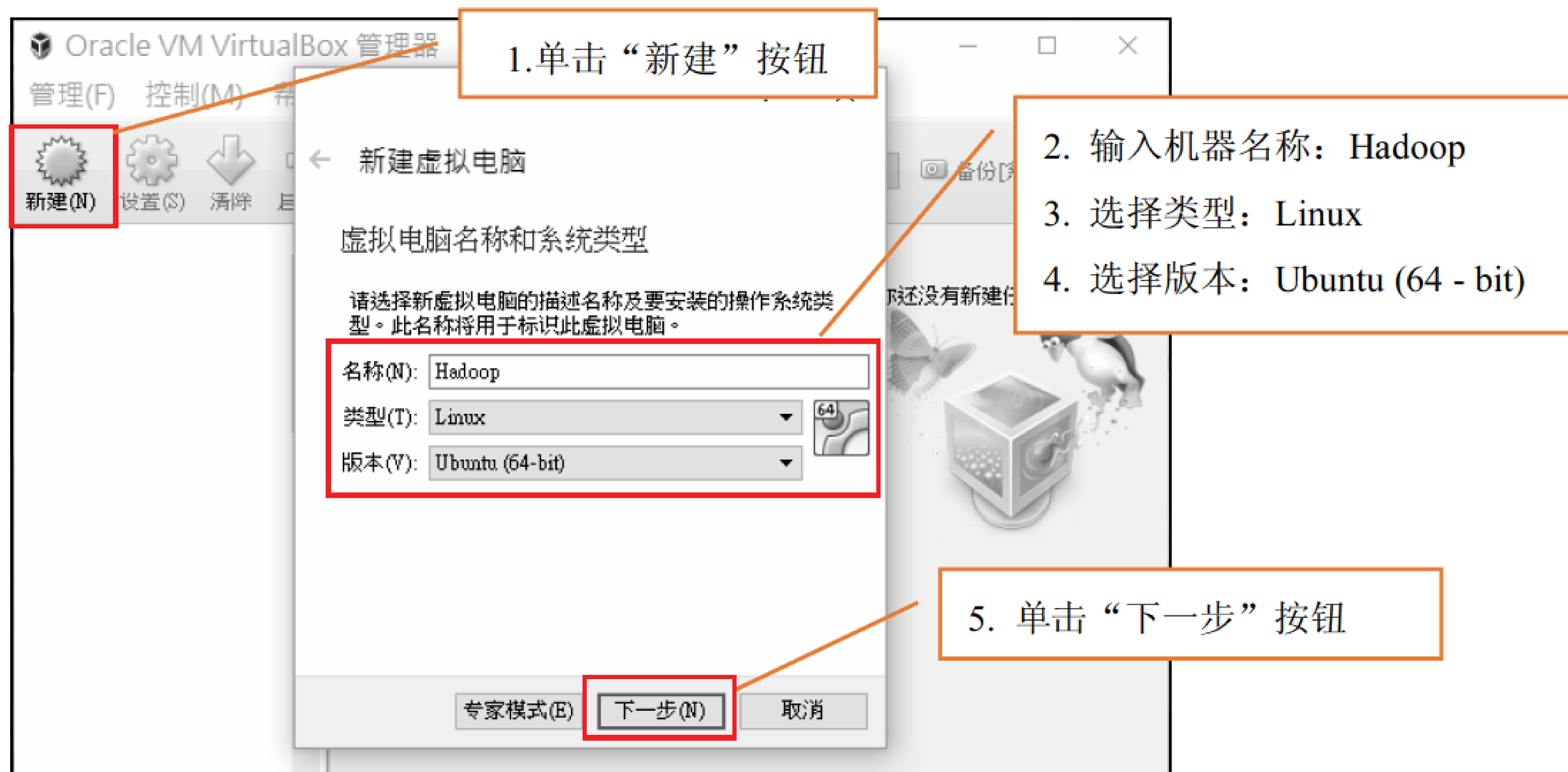


图 2-14 设置基本信息

步骤 02 设置虚拟机内存的大小（见图 2-15）

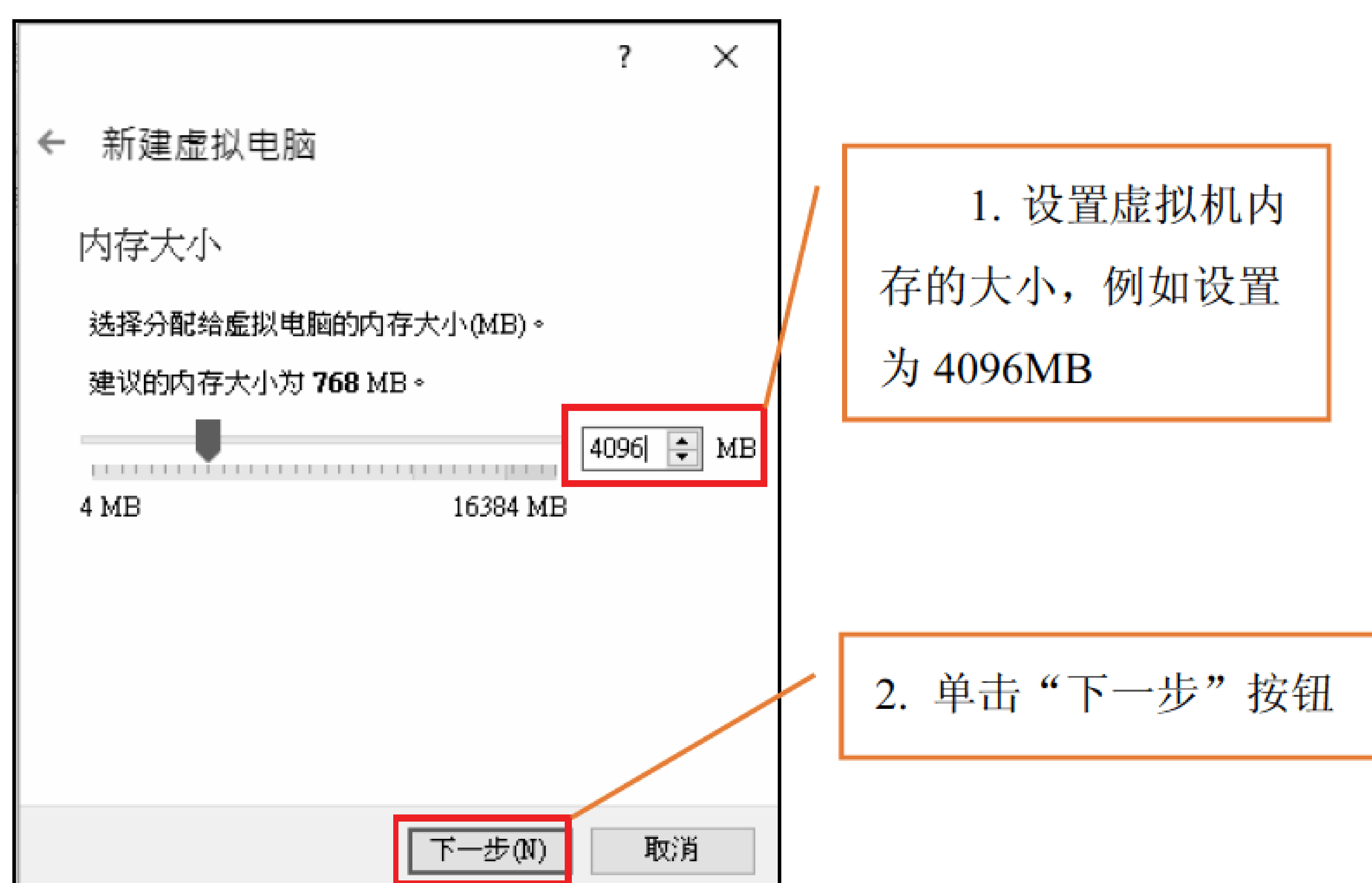


图 2-15 设置虚拟机的内存大小

步骤 03 创建虚拟机硬盘

选中“现在创建虚拟硬盘”单选按钮，再单击“创建”按钮，如图 2-16 所示。

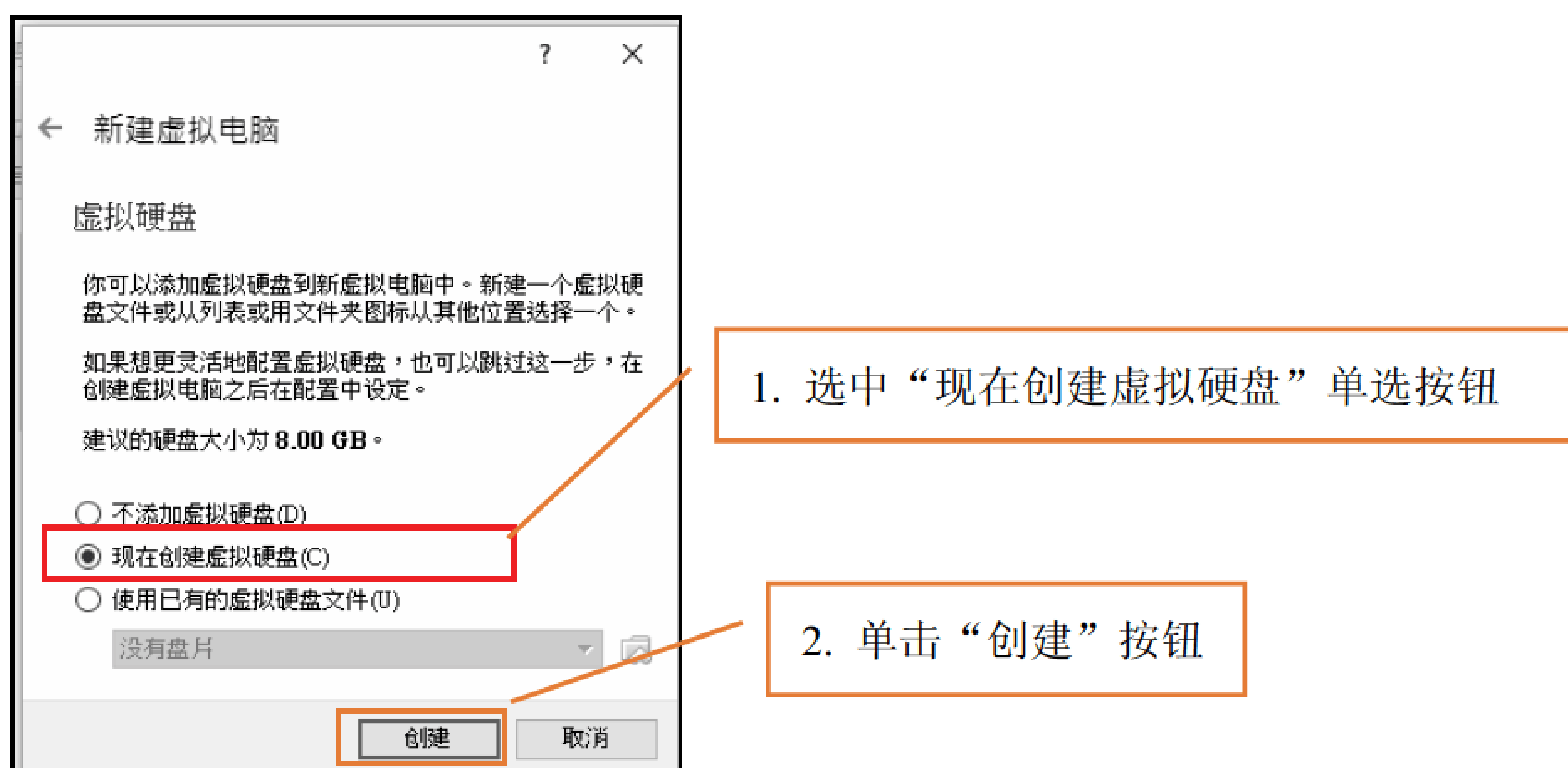


图 2-16　创建虚拟硬盘

步骤 04 选择虚拟硬盘文件类型

选择虚拟硬盘的文件格式，默认为 VDI，如图 2-17 所示。

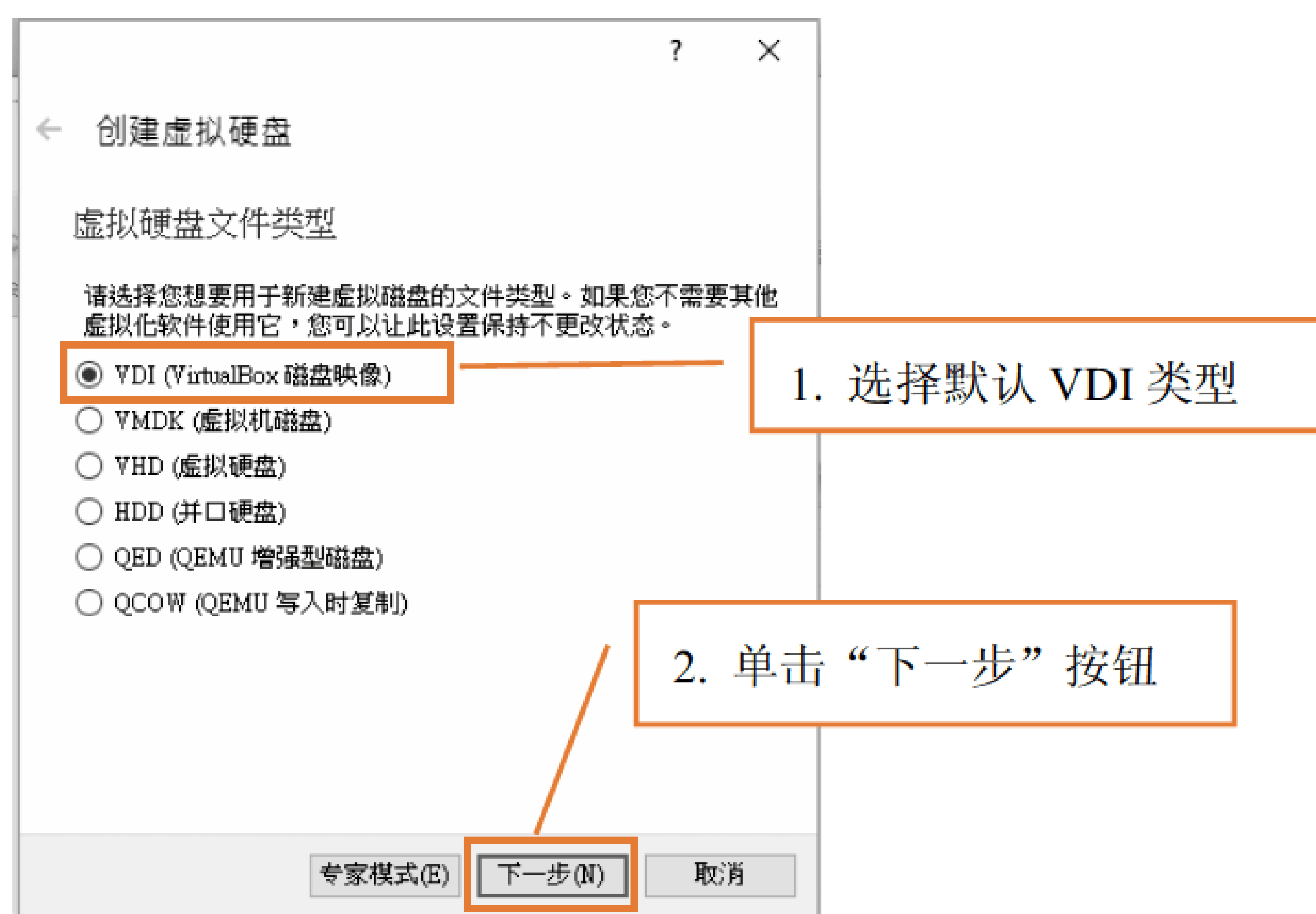


图 2-17　选择虚拟硬盘文件类型

步骤 05 设置虚拟硬盘的分配方式

设置虚拟硬盘的分配方式为“动态分配”，如图 2-18 所示。选择动态分配的好处是不用担心会占用太多硬盘空间。虚拟硬盘会随着虚拟机扩展慢慢地增加存储空间。

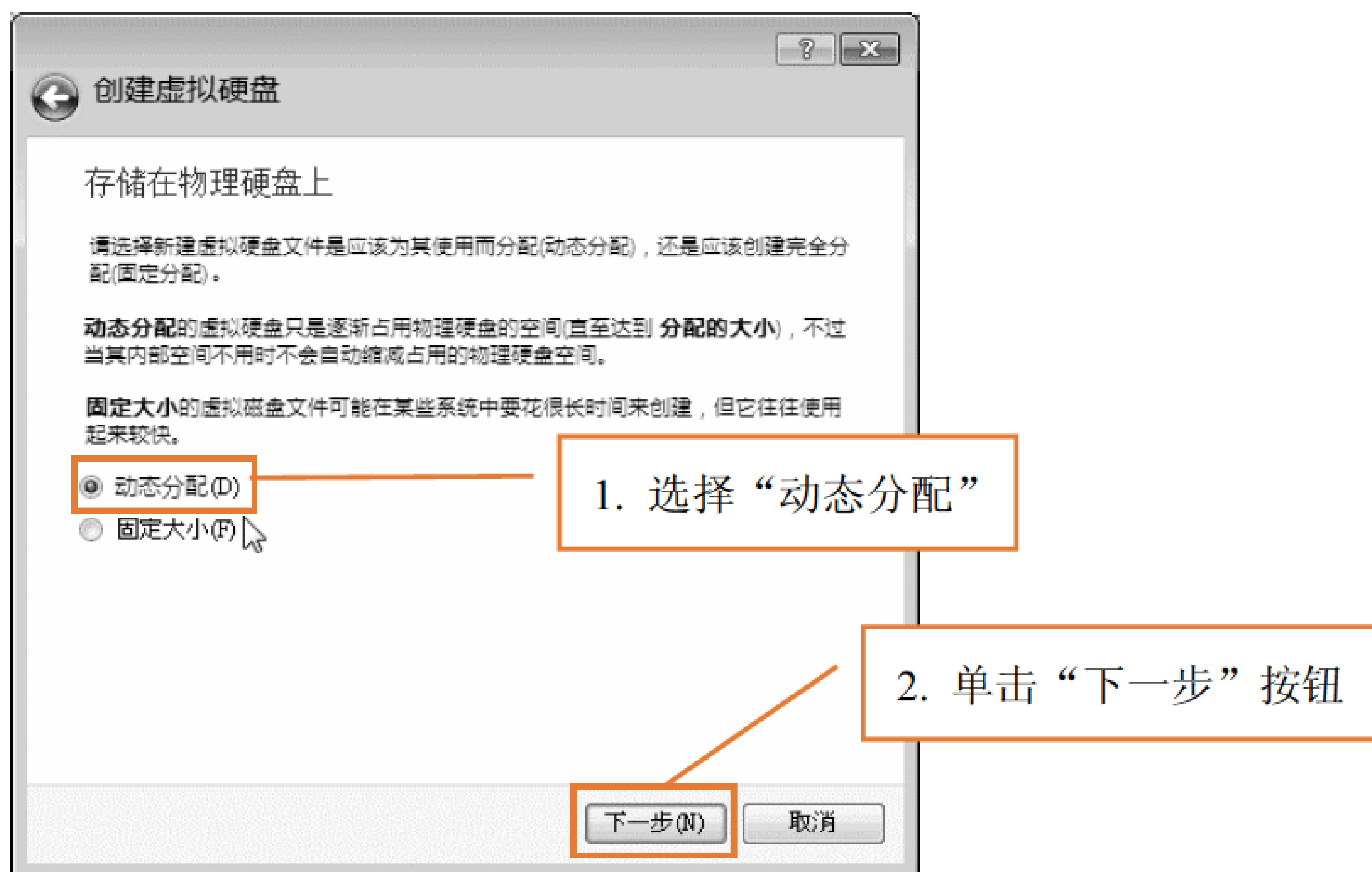


图 2-18　设置分配方式

步骤 06 设置虚拟硬盘的文件位置和大小

接下来，设置虚拟硬盘的文件位置和大小，如图 2-19 所示。文件会创建在之前所设置的默认文件夹中。文件大小设置为 100GB（这是上限值，实际文件会动态增加到上限为止）。

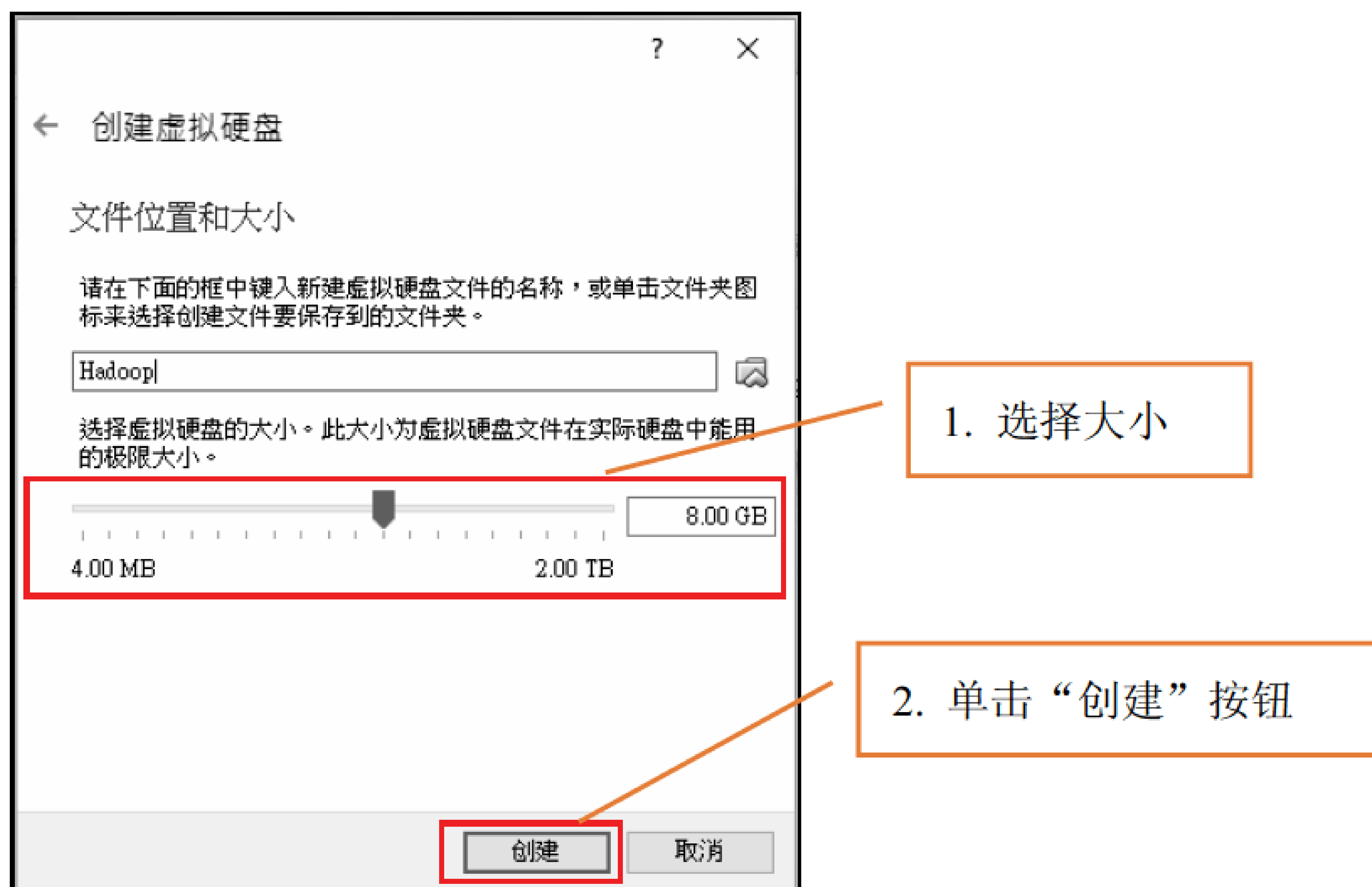


图 2-19　设置文件位置和大小

步骤 07 已经创建的虚拟机

创建完成之后，就可以看到虚拟机的图标了，如果 2-20 所示。

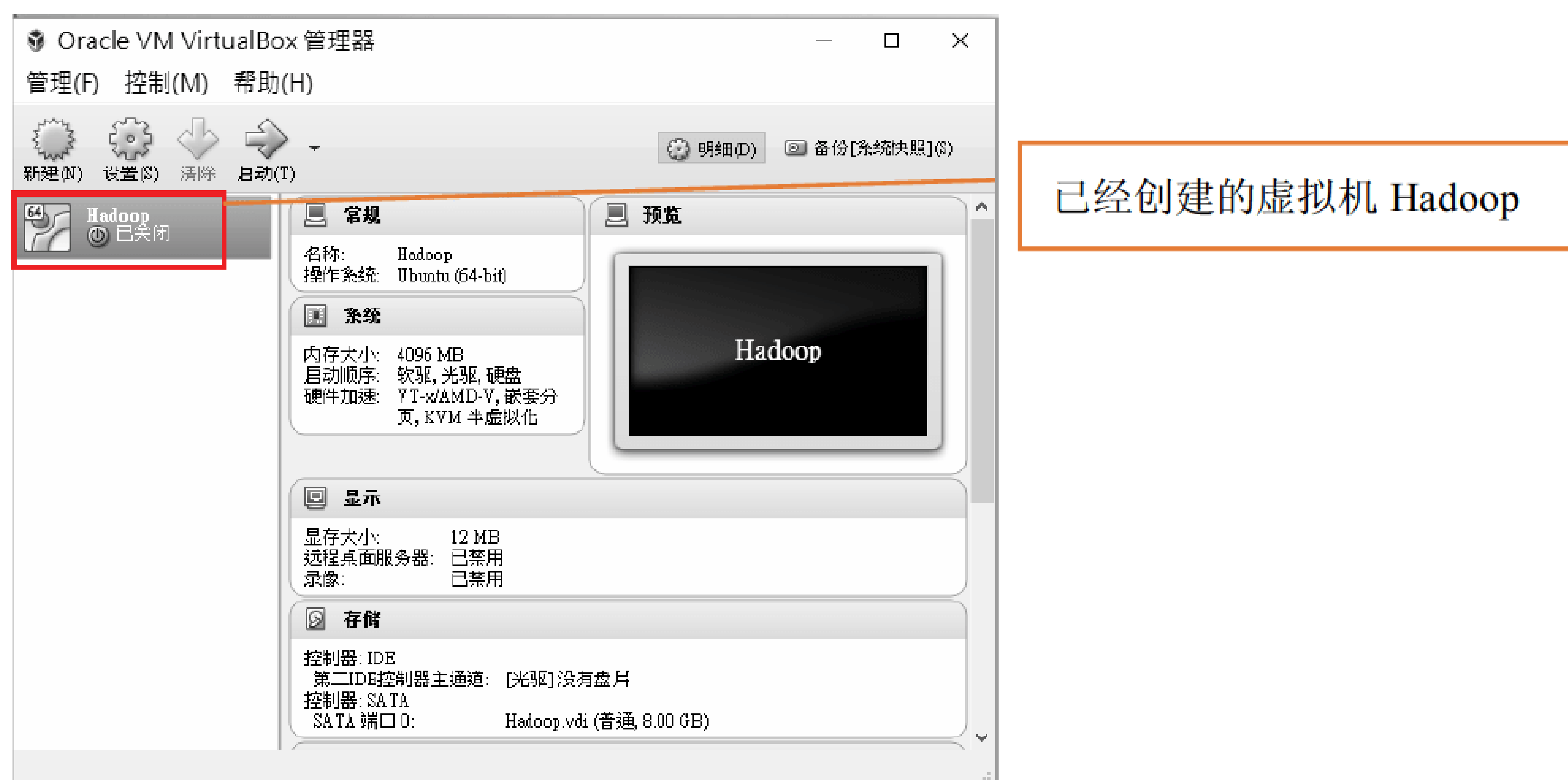


图 2-20　创建好虚拟机

步骤 08　虚拟机文件

创建完成后，在之前设置的默认虚拟机文件夹中，VirtualBox 会再帮文件创建一个子目录 Hadoop，其中有 3 个文件，即虚拟机文件，如图 2-21 所示。

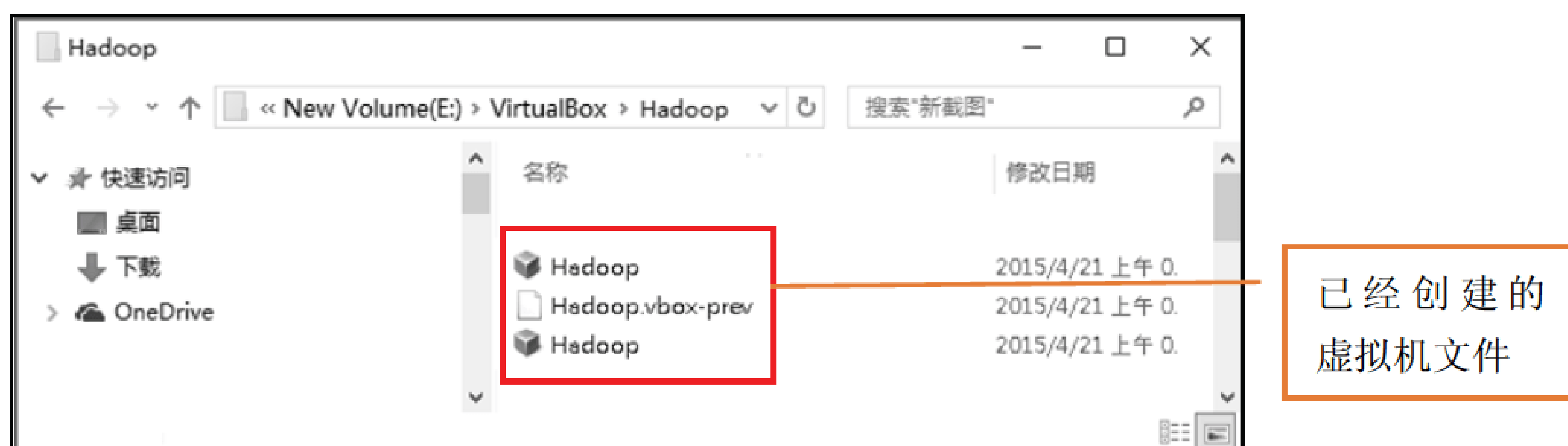


图 2-21　虚拟机文件

2.4　结论

本章我们介绍了如何安装 Virtual Box 虚拟机软件，并且创建了 Hadoop 虚拟机。下一章，我们将在 Hadoop 虚拟机上安装 Ubuntu 操作系统。

第 3 章

Ubuntu Linux 操作系统的安装

Ubuntu 是众多 Linux 操作系统版本中的一种，由 Canonical 公司维护并发行。Ubuntu 来自非洲南部一带，意思为“乐于分享”。Ubuntu 提供了 GNOME 桌面环境，是一个开放源码、功能强大而且免费的操作系统，可以自由下载、复制、使用甚至修改。除了免费的操作系统本身外，其中还包括文字数据处理、影音播放、图像处理、刻录等各种软件。

本章将介绍如何在 Virtual Box 虚拟机上安装 Ubuntu Linux 操作系统。

3.1 Ubuntu Linux 操作系统的安装

步骤 01 下载 Ubuntu 的网址

打开浏览器，输入 http://www.ubuntu.com/download/alternative-downloads 网址，连接至 Ubuntu 官网，下载安装光盘文件，如图 3-1 所示。

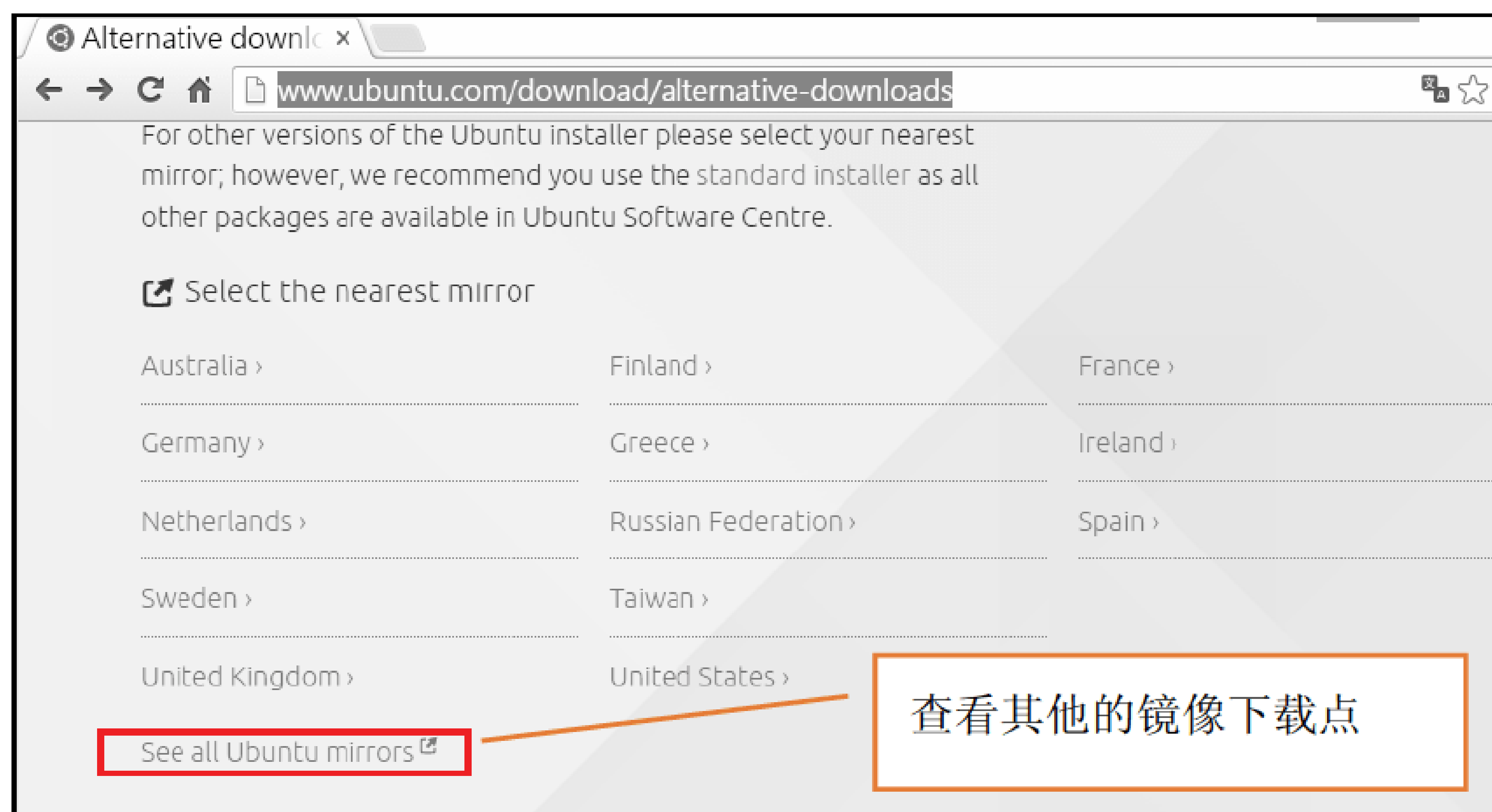


图 3-1　连接至 Ubuntu 官网

步骤 02 选择镜像（Mirror）下载点（见图 3-2）

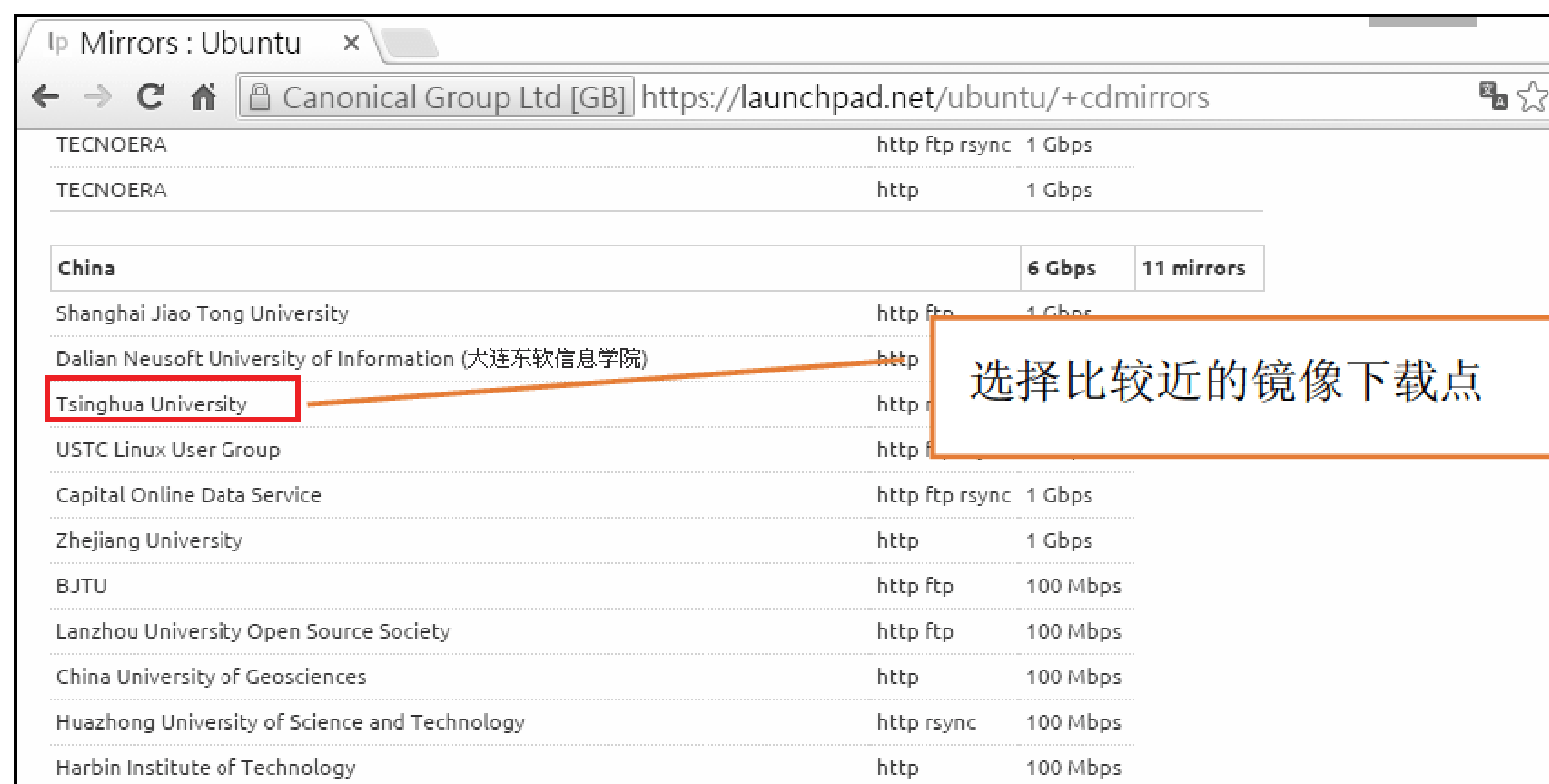


图 3-2　选择镜像下载点

步骤 03 用鼠标单击要下载 Ubuntu 的网址（见图 3-3）

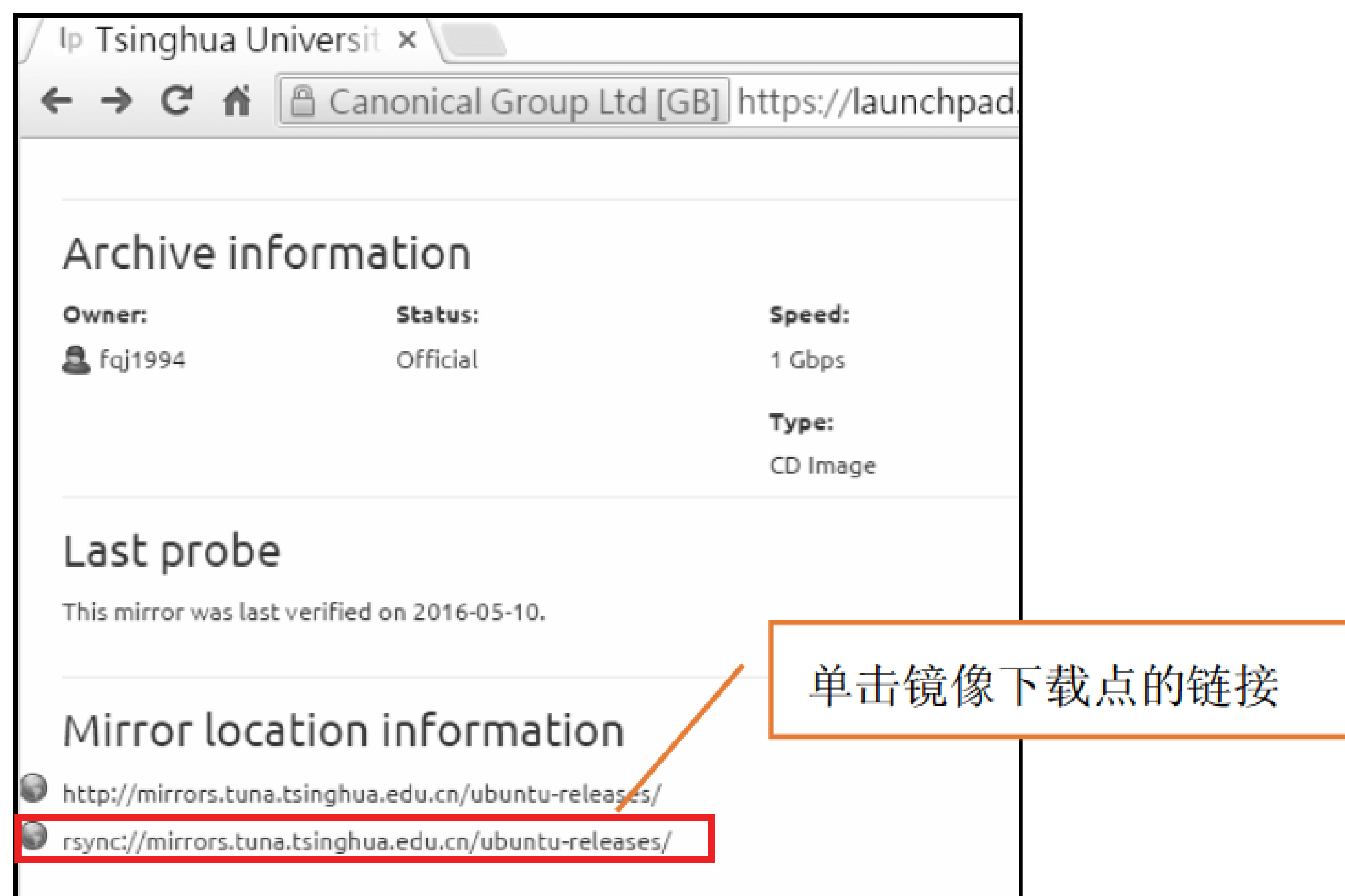


图 3-3　单击链接

步骤 04 用鼠标单击选择 14.04 版本（见图 3-4）

本书选择安装 14.04 版本的 Ubuntu，因为本书命令与范例程序都已经在此版本测试无误。

LTS（Long Term Support）是长期支持的版本。Ubuntu 对台式机（Desktop）版本提供 3 年、服务器（Server）版提供 5 年的更新支持，因为企业必须有稳定的环境，每次的版本升级都可能是涉及软硬件配合的大工程，并且不太可能常常升级版本。虽然不升级，但是仍然必须要更新，所以选择 LTS 版本。

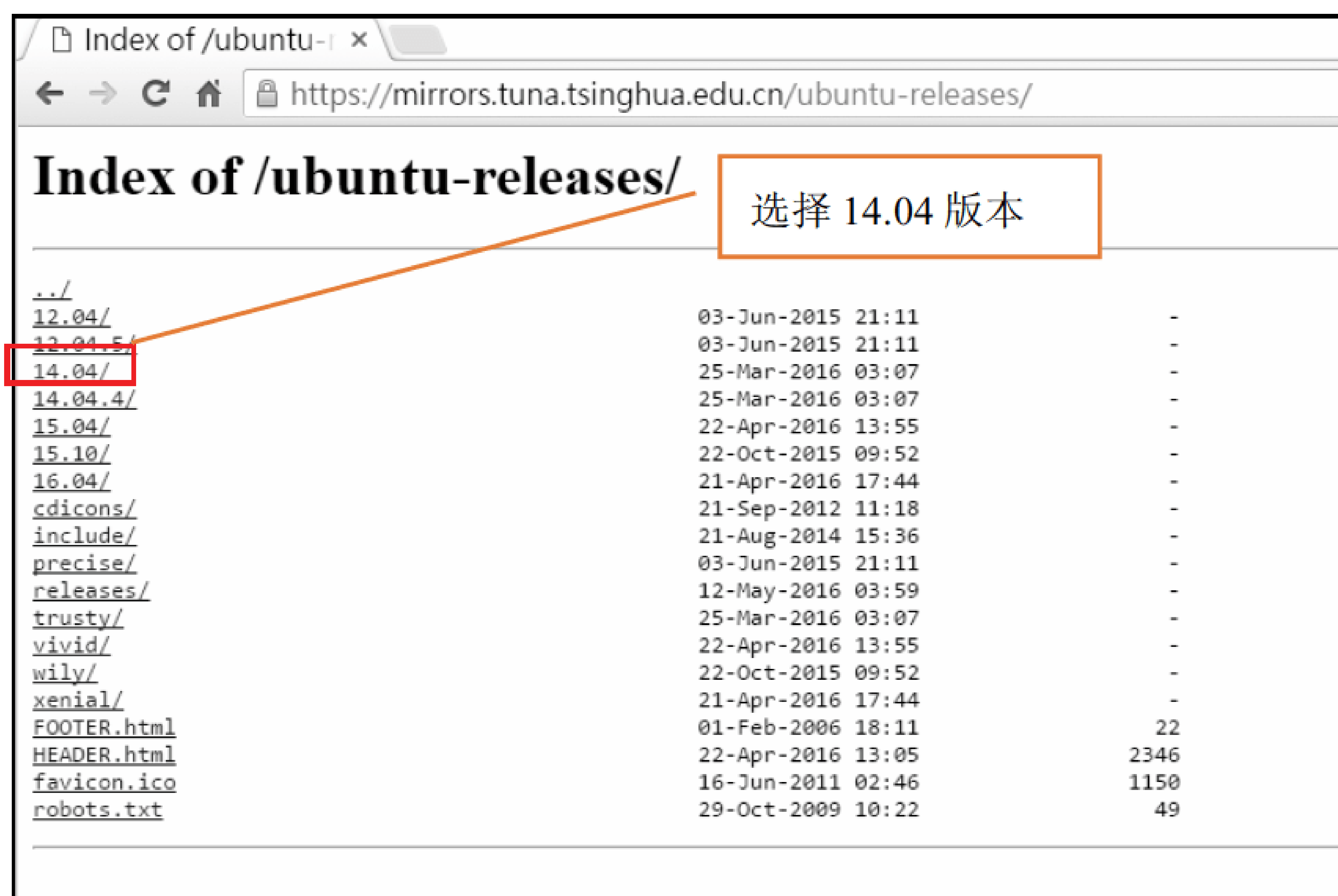


图 3-4　选择版本

步骤 05 下载 14.04 版本的 iso 文件（见图 3-5）



图 3-5　下载文件

3.2 在 Virtual 设置 Ubuntu 虚拟光盘文件

在物理计算机中安装软件时就是直接把光盘片放到光驱中进行安装，在虚拟机中要安装软件时则需要设置虚拟光盘文件，告诉虚拟机安装光盘在哪里，这样虚拟机才能读取到安装光盘文件。

接下来，设置之前下载的 Ubuntu 虚拟光盘文件，告诉虚拟机安装光盘在哪里。

步骤 01 设置虚拟光盘文件（见图 3-6）

按照下列步骤设置虚拟光盘文件。

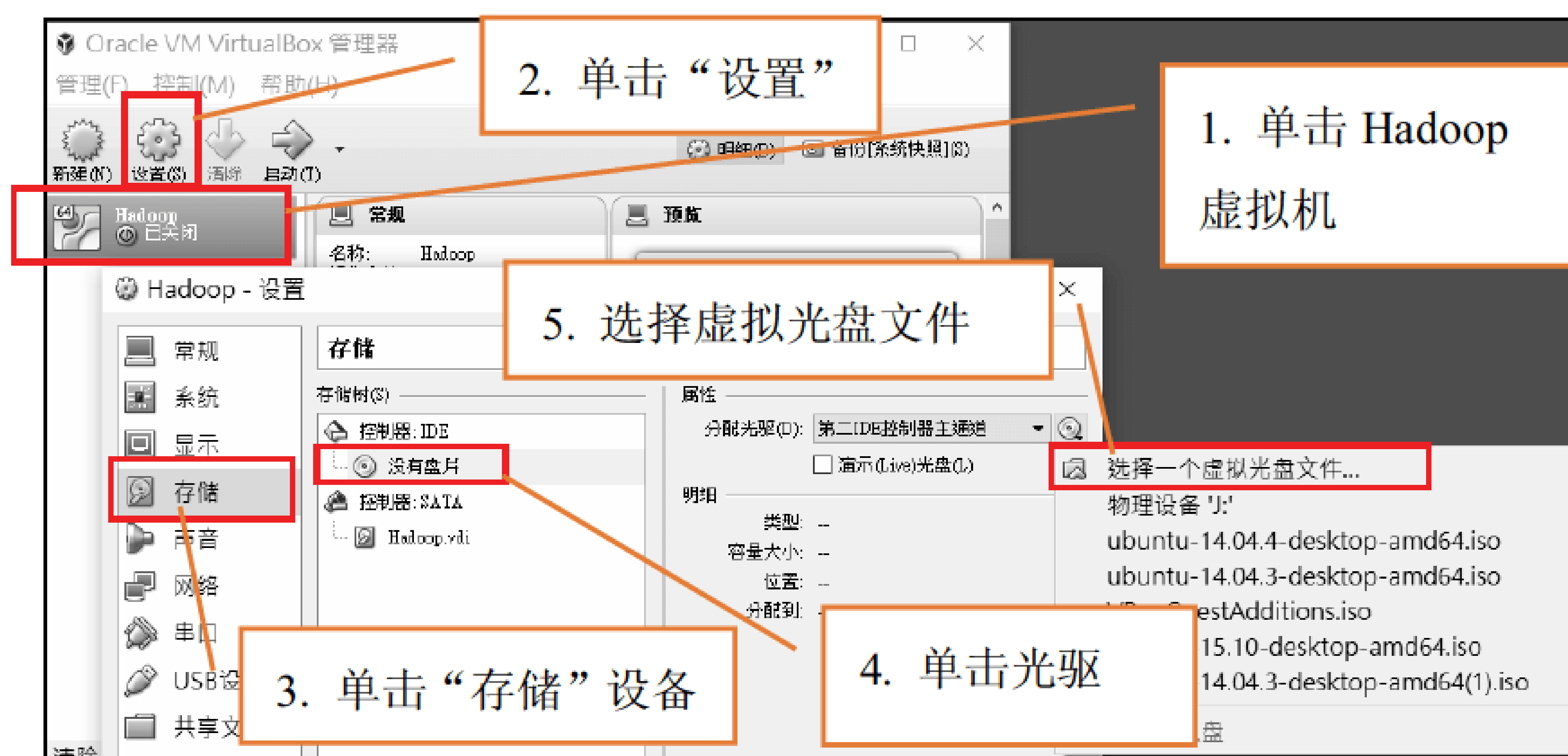


图 3-6　设置虚拟光盘文件

步骤 02 选择 Ubuntu 安装光盘文件（见图 3-7）

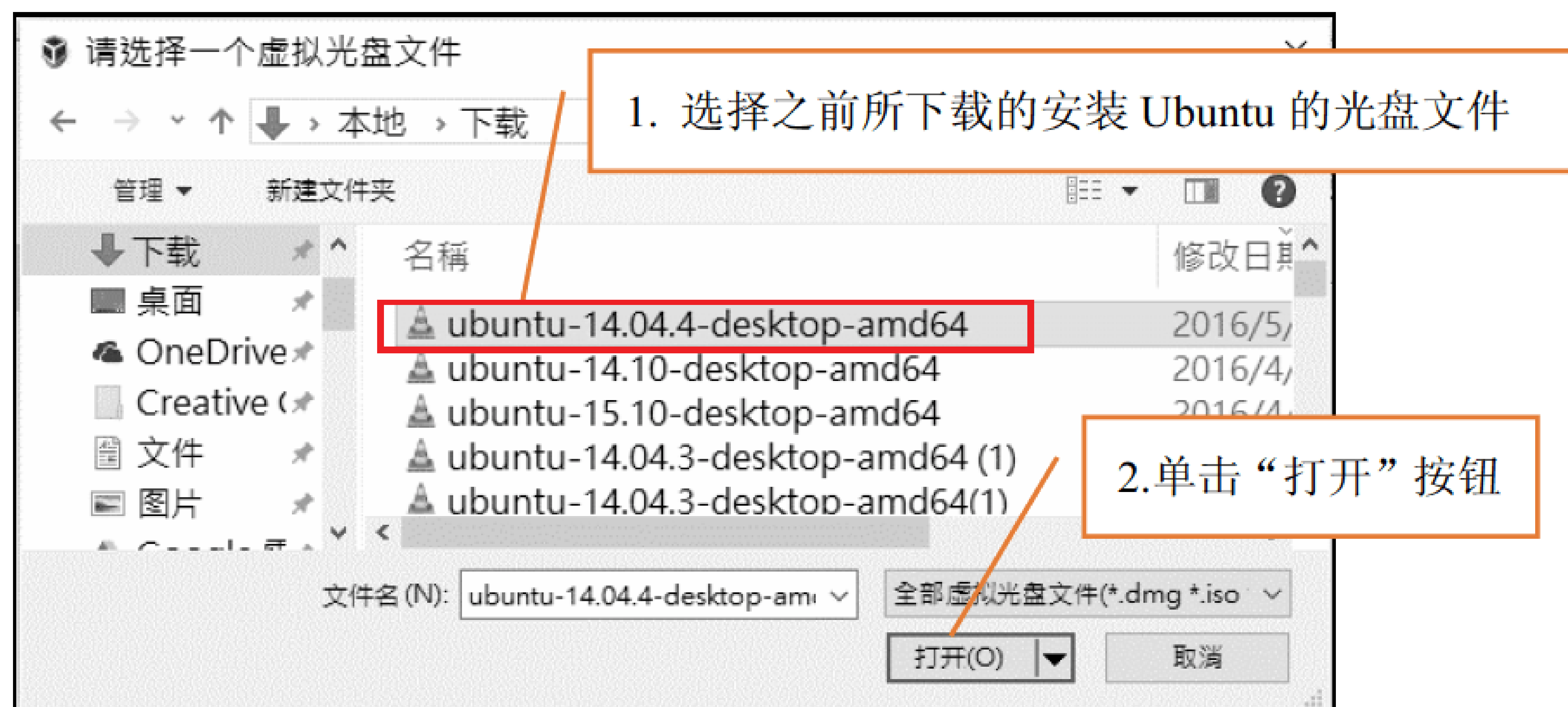


图 3-7　选择安装光盘文件

步骤 03　选择虚拟光盘文件（见图 3-8）

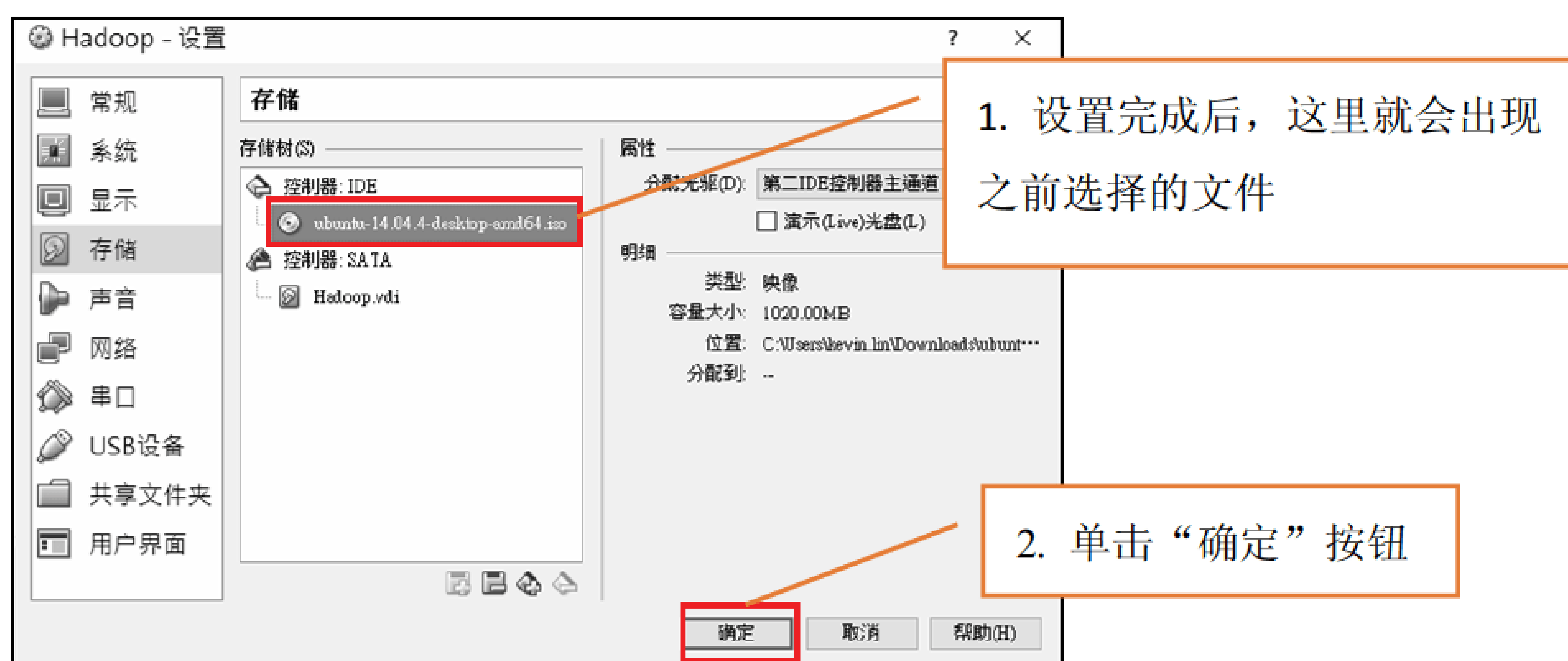


图 3-8　选择虚拟光盘文件

步骤 04　完成虚拟光盘文件的设置（见图 3-9）



图 3-9　完成设置

3.3　开始安装 Ubuntu

在之前的章节中，我们已经设置了安装 Ubuntu 的光盘文件。现在我们要启动虚拟机。

步骤 01　启动虚拟机（见图 3-10）



图 3-10　启动虚拟机

步骤 02　选择安装 Ubuntu 语言版本

接下来选择安装 Ubuntu 语言版本，默认为英文。你可以按键盘上的↑、↓箭头键选择安装 Ubuntu 语言版本，在此我们选择“中文(简体)”，如图 3-11 所示。



图 3-11　选择语言

步骤 03　选择是否安装更新与安装第三方软件（见图 3-12）

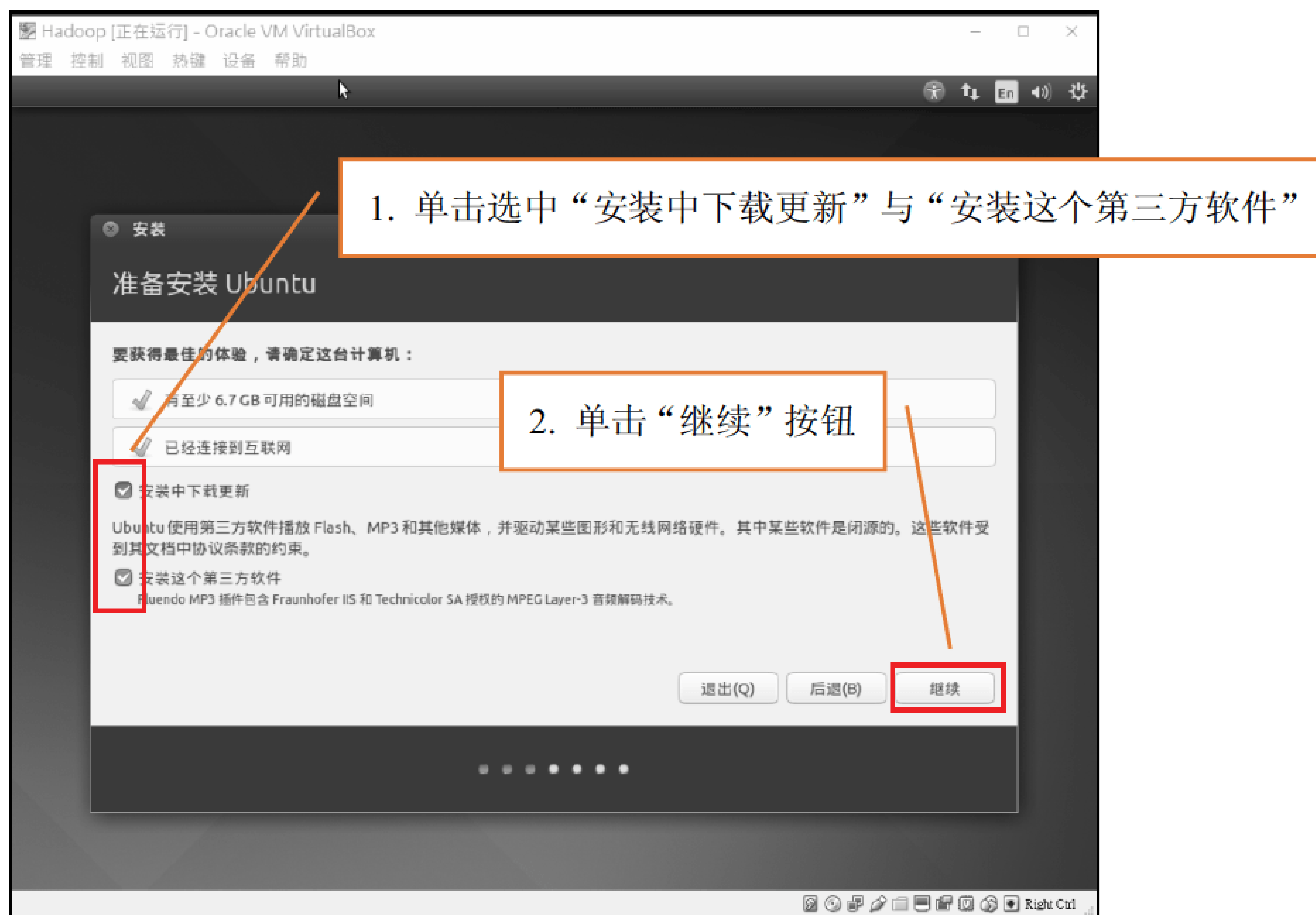


图 3-12　选择安装更新与第三方软件

步骤 04 选择“清除整个磁盘并安装 Ubuntu”（见图 3-13）



图 3-13　清除整个磁盘并安装

步骤 05 是否“将改动写入磁盘”（见图 3-14）



图 3-14　确认将改动写入磁盘

步骤 06 选择机器所在地（见图 3-15）



图 3-15　选择机器所在地

步骤 07 选择机器所在地完成（见图 3-16）



图 3-16　继续操作

步骤08 选择键盘布局方式（见图 3-17）

请特别注意键盘布局方式是“英语（美国）”，而不是“汉语（SunPinyin）”。



图 3-17 选择键盘布局方式

步骤09 设置姓名、计算机名，选择一个用户名并设置账号密码（见图 3-18）

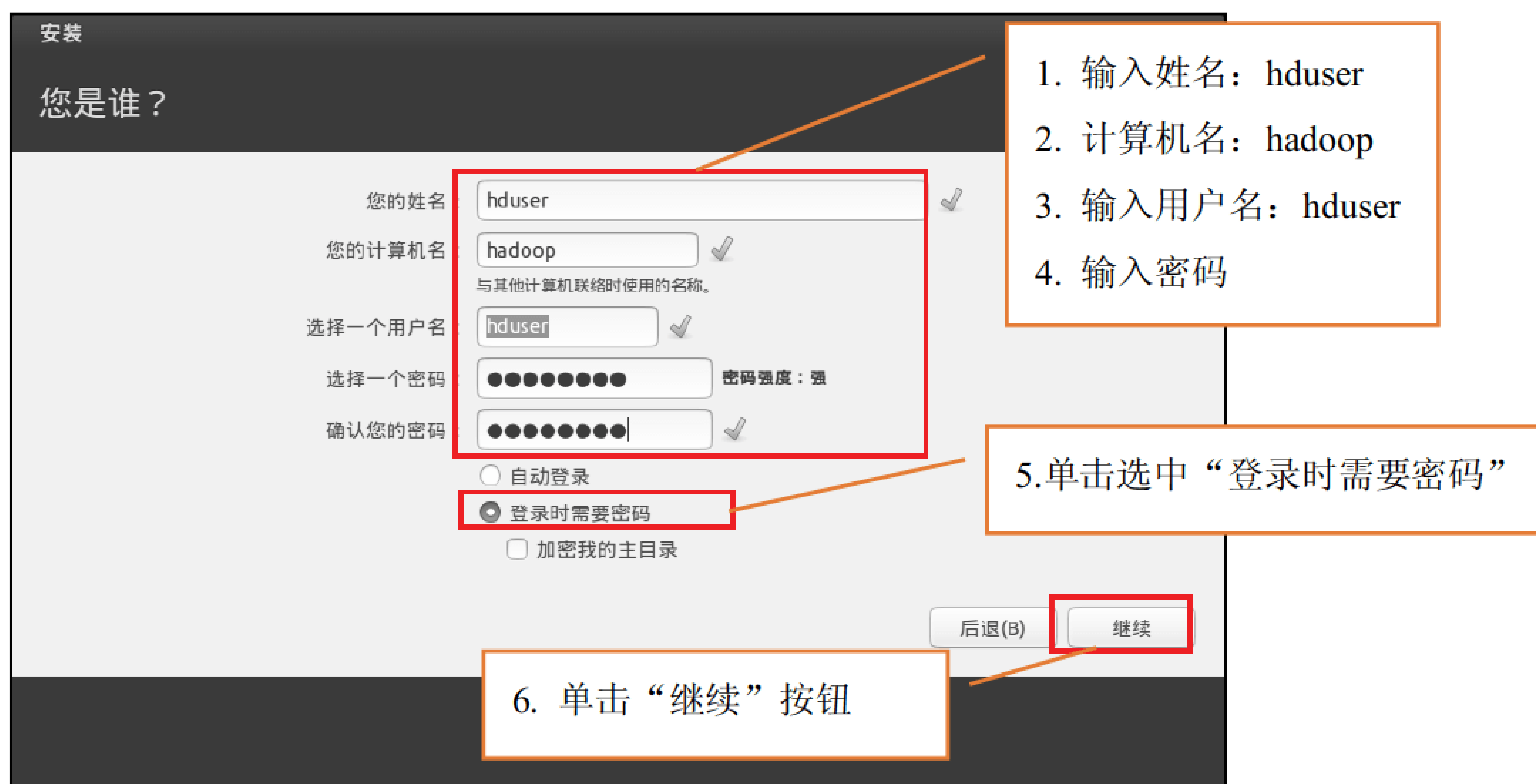


图 3-18 设置登录项

步骤10 开始安装

开始安装时的屏幕显示界面如图 3-19 所示。



图 3-19　开始安装

步骤 11　安装完成并选择重新启动（见图 3-20）



图 3-20　选择重新启动

步骤 12　重新启动（见图 3-21）

重新启动时，等候一段时间后界面不再改变。如果系统没有自动退出，可以将其强制退出。

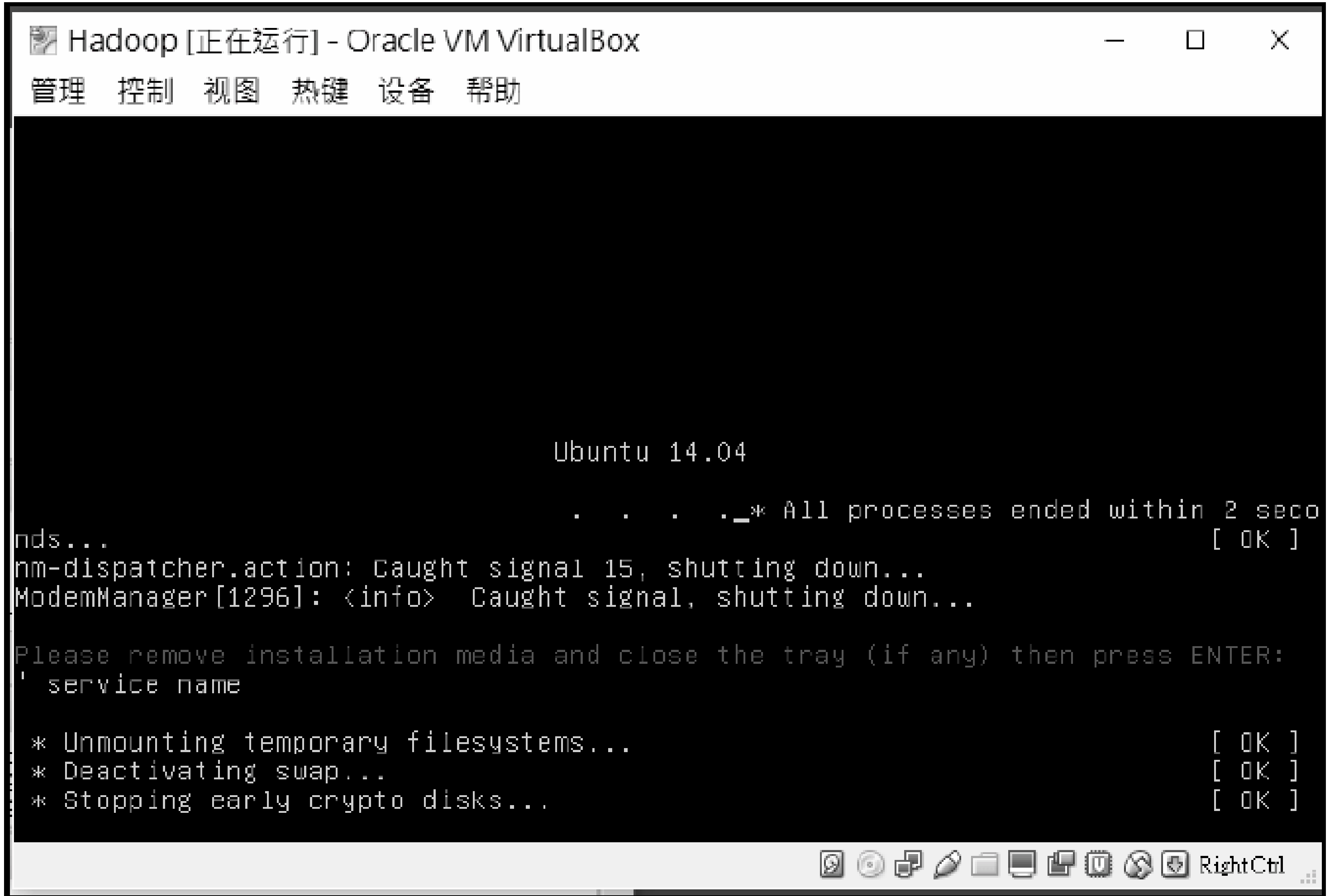


图 3-21　重新启动

步骤 13　强制退出

强制退出的方法如图 3-22 所示。

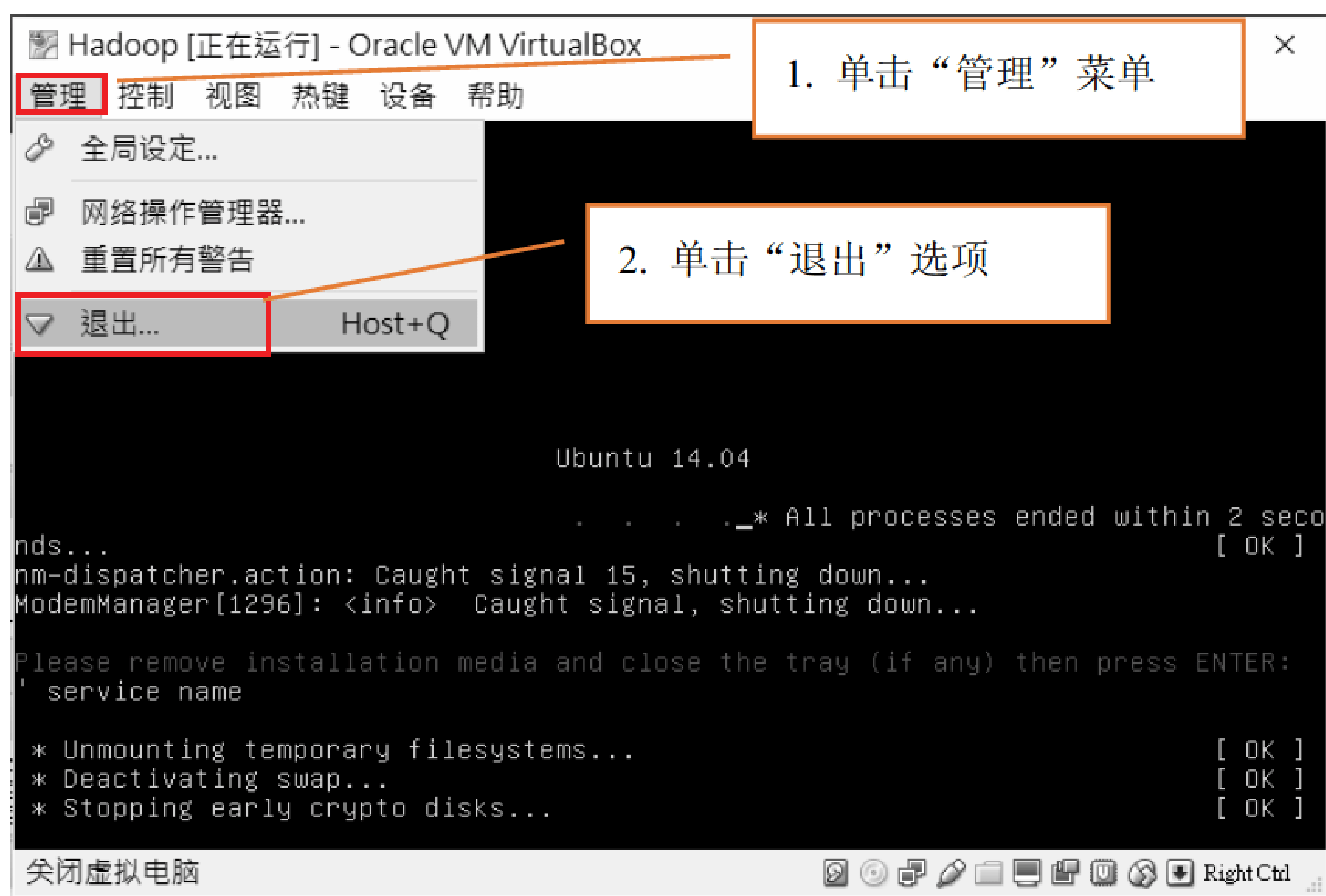


图 3-22 强制退出菜单操作

步骤 14 退出（见图 3-23）

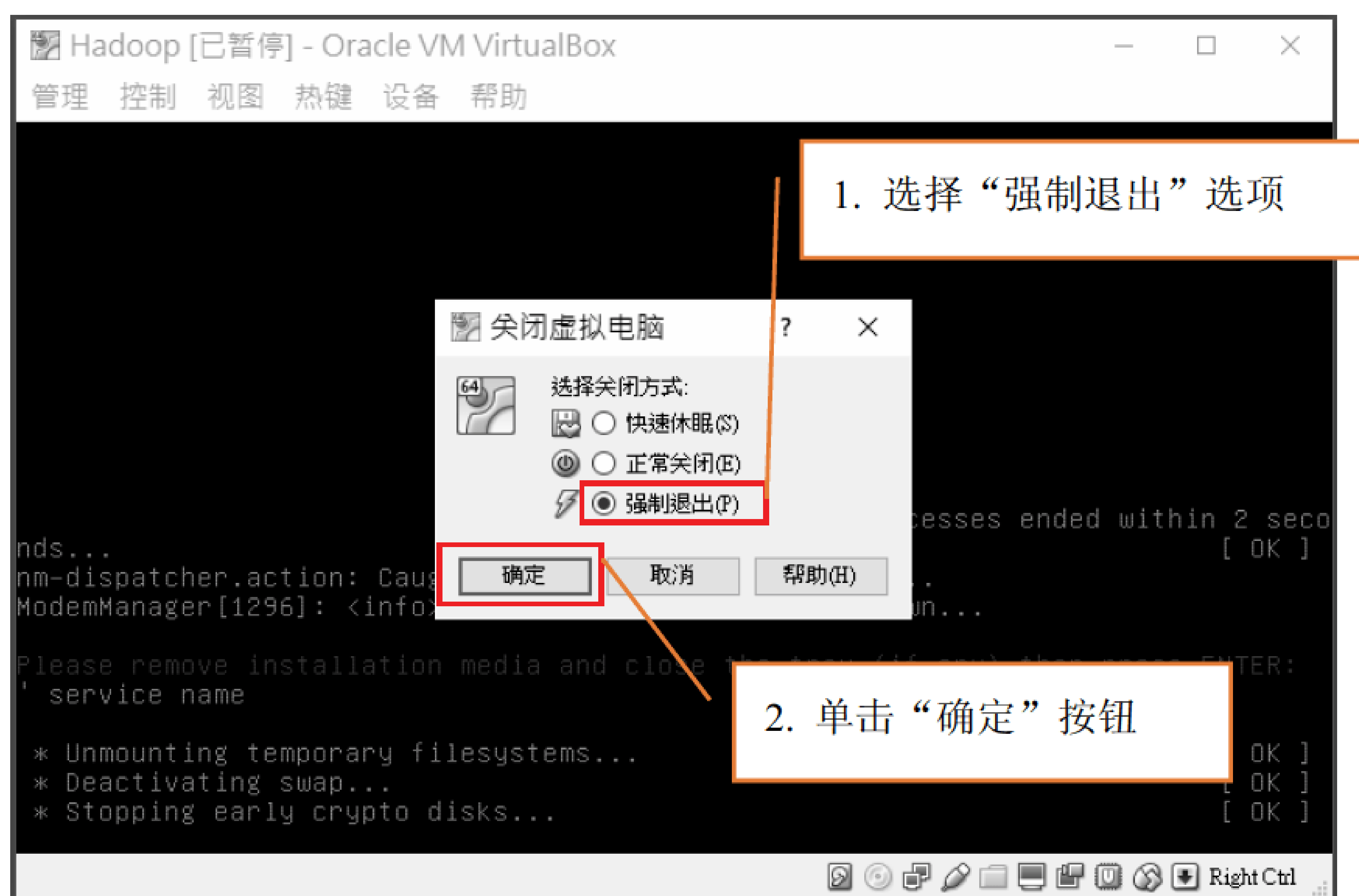


图 3-23 退出

3.4 启动 Ubuntu

在之前的步骤中我们已经完成了 Ubuntu 的安装，现在要开机了。

步骤 01 启动 Ubuntu（见图 3-24）

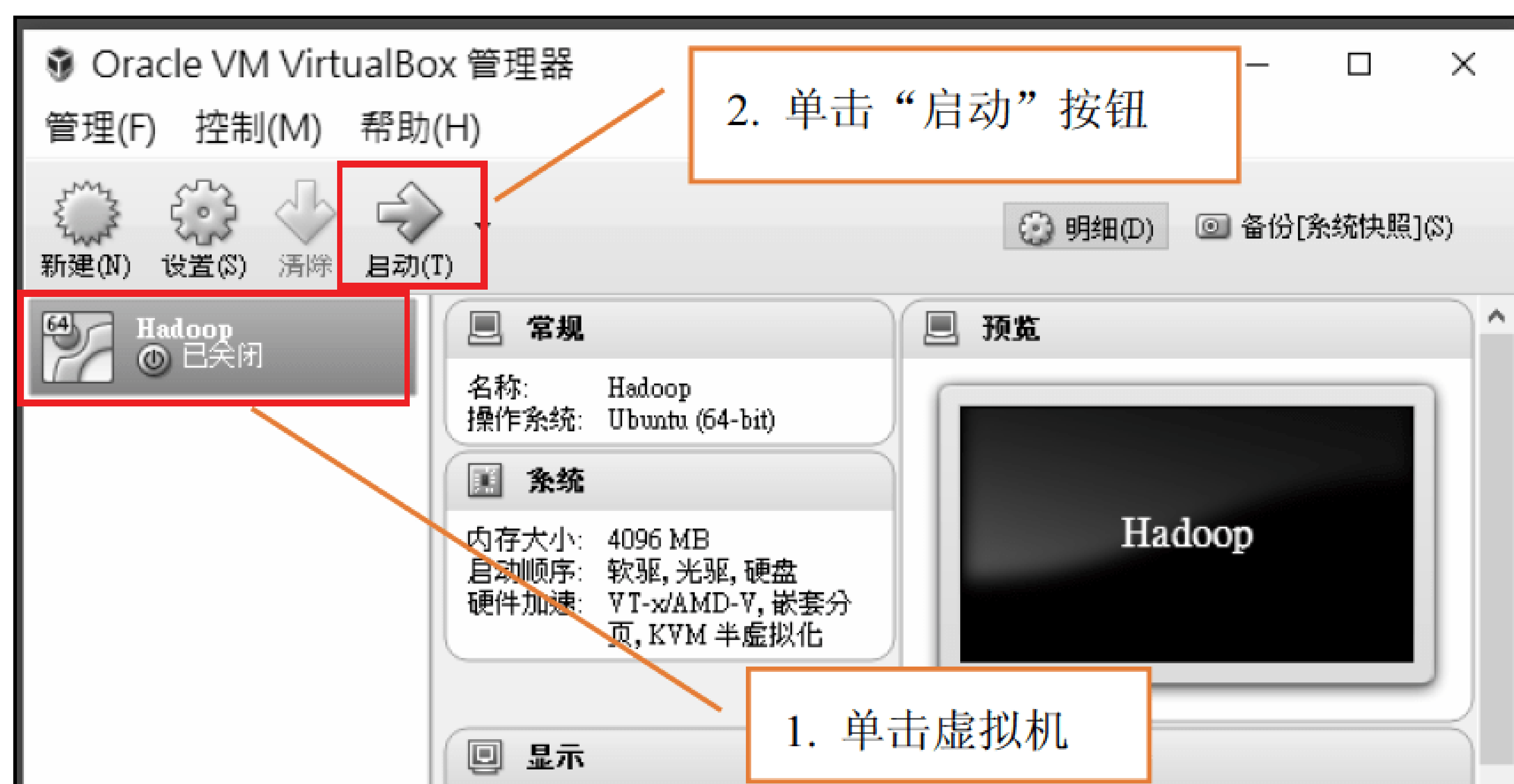


图 3-24 启动 Ubuntu

步骤 02 输入密码（见图 3-25）



图 3-25 输入密码

重新启动虚拟机后，系统会要求输入之前设置的密码，然后按 Enter 键即可。



如果输入密码一直不成功，而且输入时文字有下划线，就有可能是系统自动切换至中文输入法了，从而无法输入密码。可以先按 Ctrl + Shift 组合键切换输入法为英文，再输入密码。

3.5 安装增强功能

Ubuntu 安装基本上已经完成了，只是还有一些问题：

- 屏幕分辨率不够。
- 鼠标光标停顿延迟。
- 无法与原安装系统共享剪贴板。

这些问题必须安装增强功能（Guest Additions）才能解决，本节将介绍安装增强功能的步骤。

步骤 01 插入 Guest Additions 光盘

在菜单中依次选择“设备”→“安装增强功能”，如图 3-26 所示。注意：有的系统会显示“插入 Guest Additions CD 映像”，而不是“安装增强功能”，两者是一个意思。



图 3-26 选择菜单项

步骤 02 运行 Guest Additions CD 光盘（见图 3-27）



图 3-27 运行

步骤 03　输入超级用户密码

因为安装 Guest Additions CD 光盘程序必须具有超级用户的权限，所以要输入之前设置的密码（提示：输入密码时请确认当前输入模式是中文还是英文，如果是中文，请按 Ctrl + Shift 组合键切换回英文模式，以免输入密码错误），如图 3-28 所示。

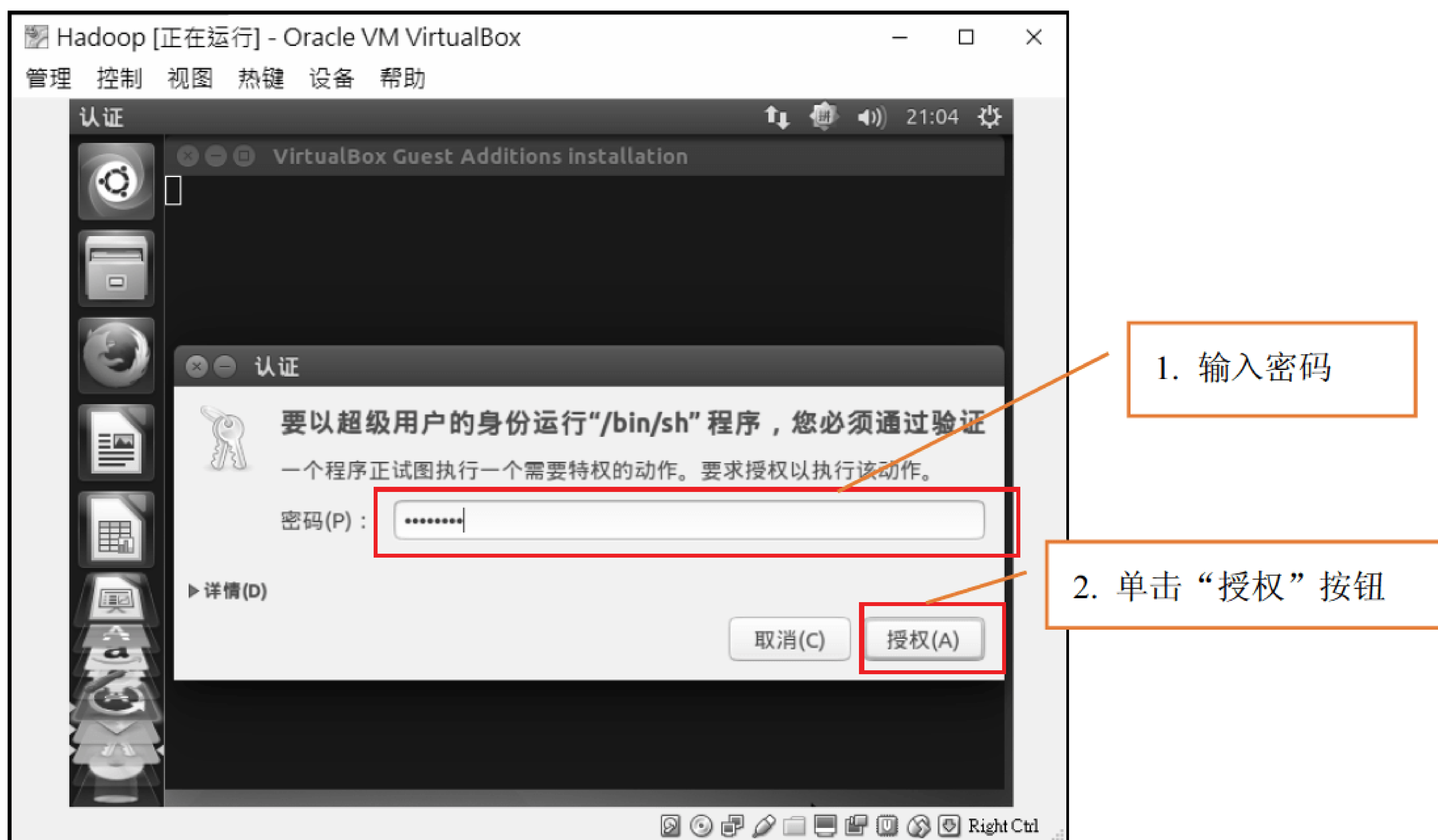


图 3-28　输入超级用户密码

步骤 04　若密码输入成功，则会出现安装 Guest Additions CD 光盘程序的界面。

出现“Press Return to close this window”代表已经安装完成（见图 3-29）。

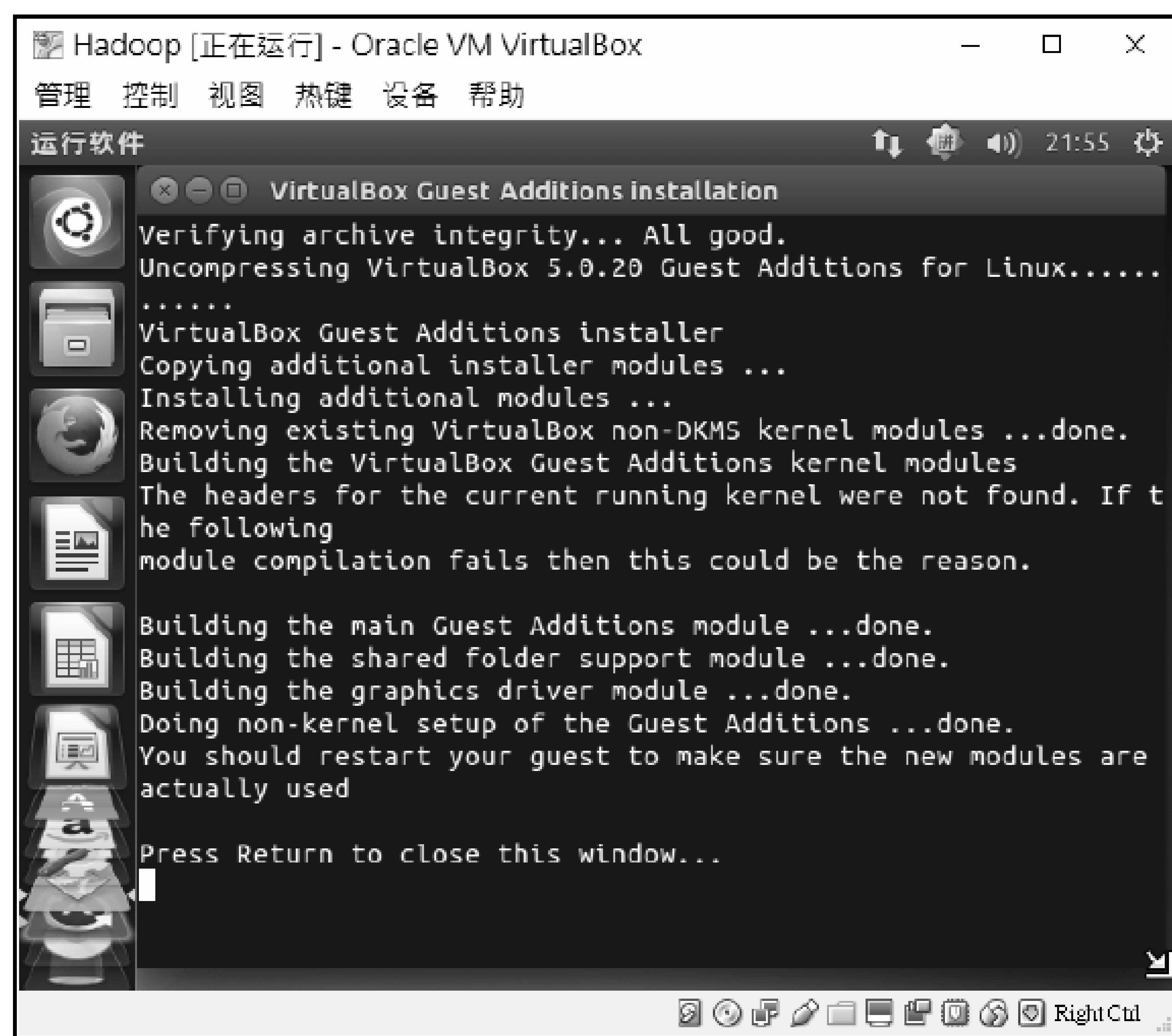


图 3-29　安装完成

步骤05 关机（见图 3-30）

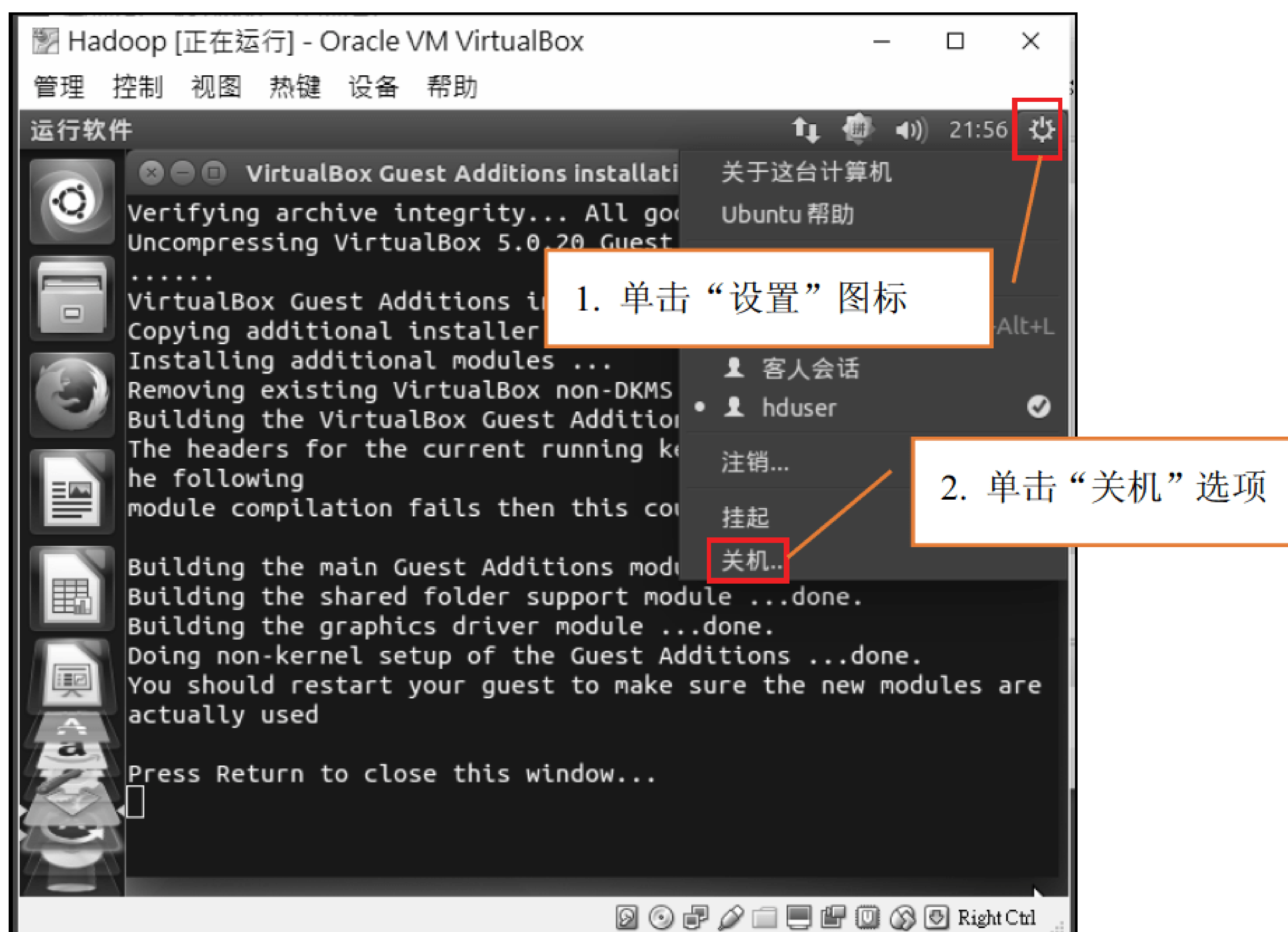


图 3-30 关机

步骤06 最后确认"关机"或"重新启动"（见图 3-31）

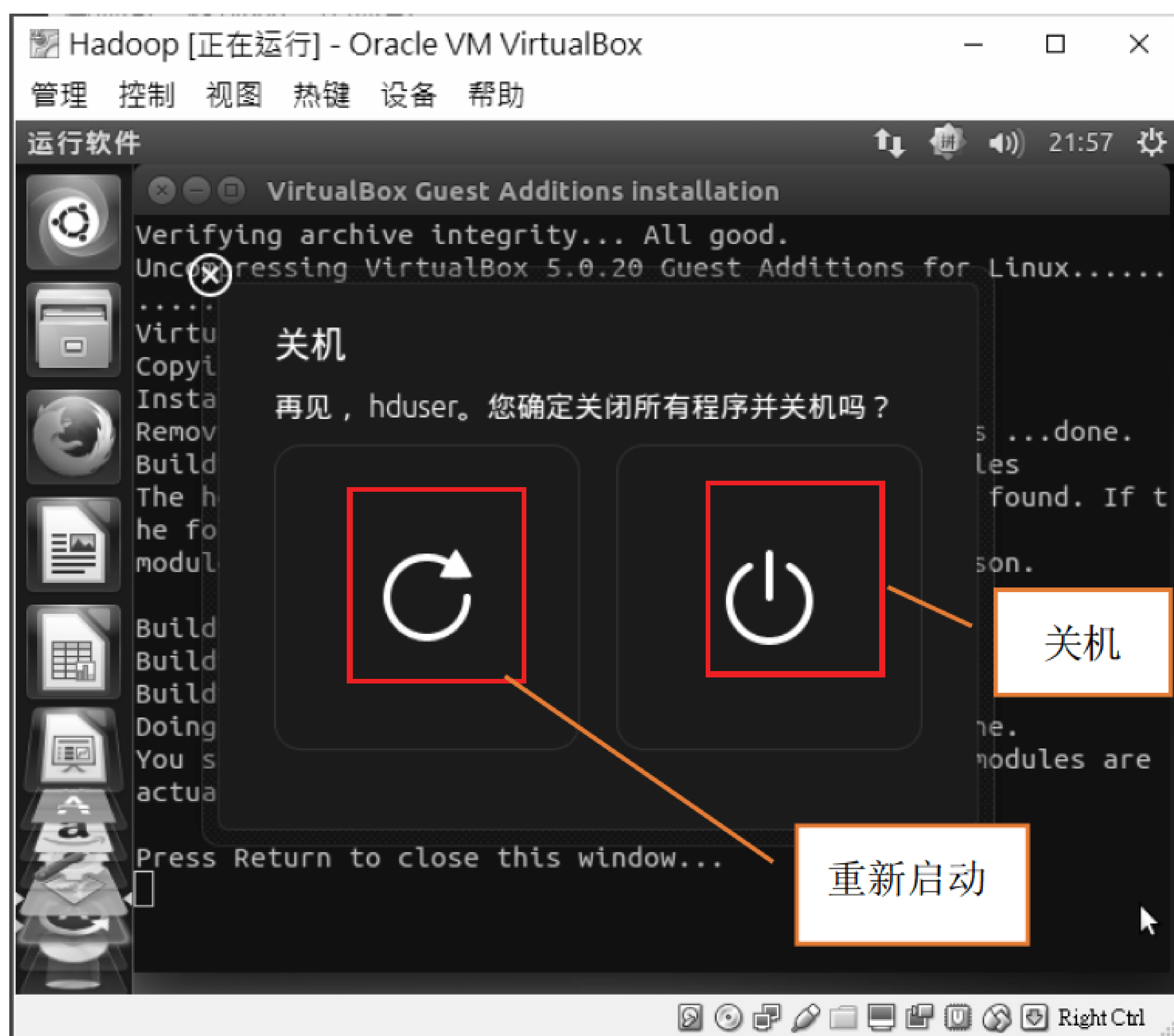


图 3-31 确认关机或重新启动

重新启动后，你会发现屏幕分辨率不够、鼠标光标停顿延迟的问题已经解决，但是仍无法与原安装系统共享剪贴板。共享剪贴板的方法会在后续章节中进行介绍。

3.6 设置默认输入法

在 Ubuntu 需要输入文字的地方默认使用的输入法是中文，可是我们大部分情况都是输入命令或程序，这些都是英文，必须自行切换输入法，这样很不方便，所以最好将默认输入法改为英文。

步骤 01 系统设置

请参照图 3-32 所示的方法打开“系统设置”窗口。



图 3-32　系统设置

步骤 02 文本输入窗口（见图 3-33）

打开“文本输入”窗口，默认的文本输入是“汉语(SunPinyin)”。

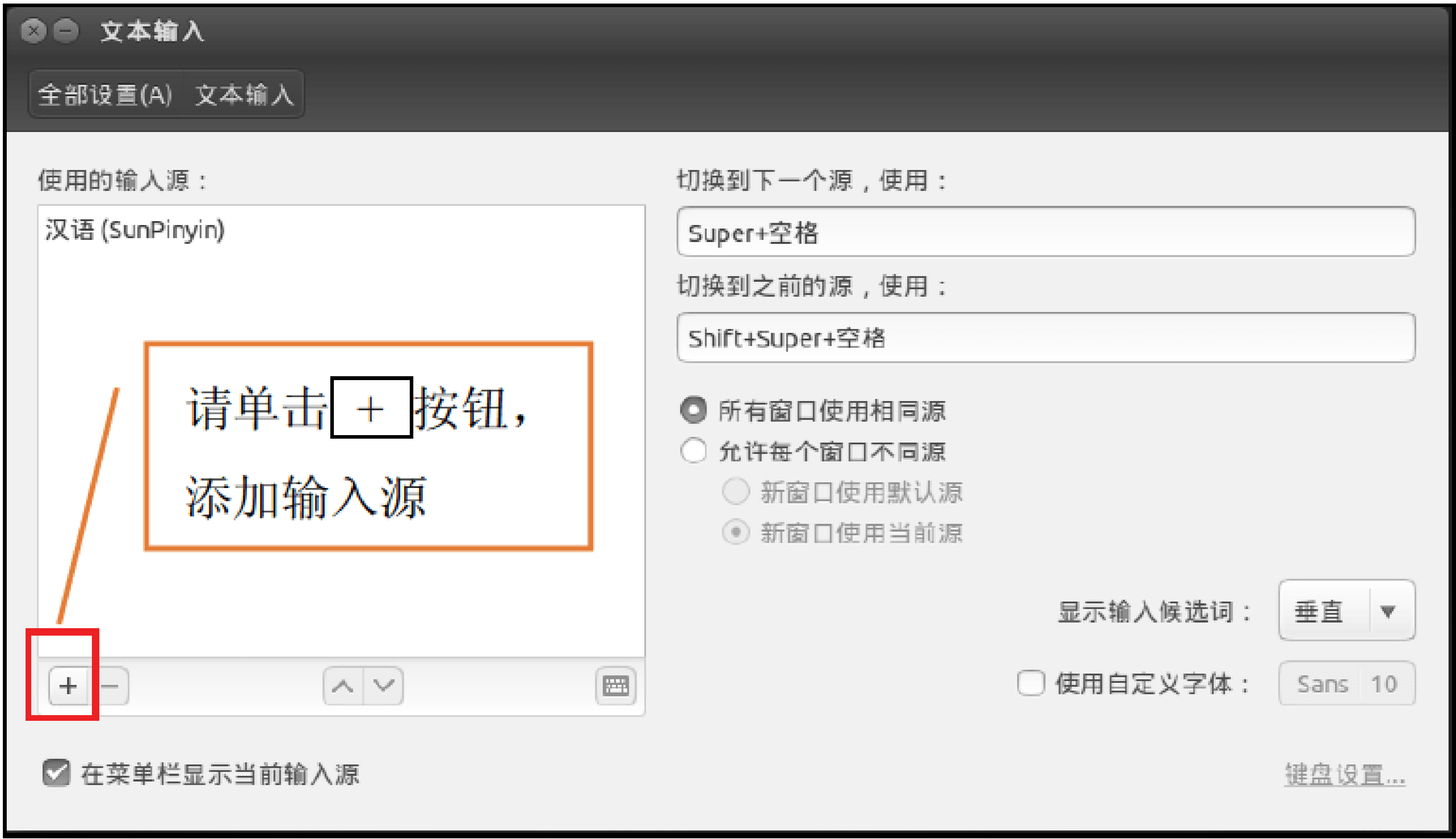


图 3-33　文本输入

步骤 03 在“选择一个输入源”对话框中选择输入源（见图 3-34）

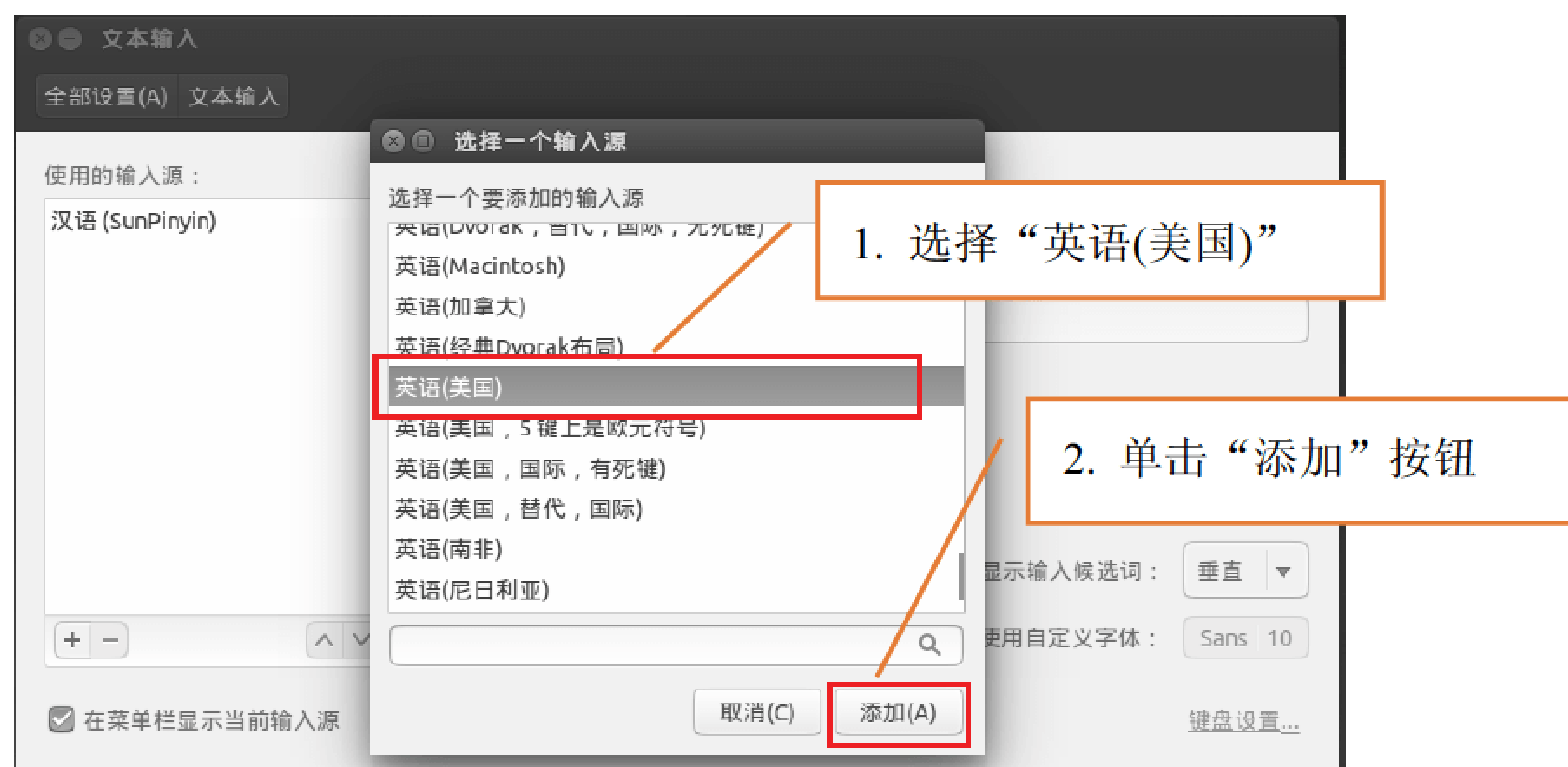


图 3-34　选择输入源

步骤 04 调整输入源的顺序（见图 3-35）



图 3-35　调整输入源的顺序

步骤 05 设置默认快捷键（见图 3-36）

切换下一个输入法的默认快捷键是 Super + 空格，如图 3-36 所示。Ubuntu 系统中的 Super 键对应到 Windows 系统就是 键。这与我们在 Windows 中的使用习惯不符，可以参考图 3-37、图 3-38 来设置快捷键。



图 3-36　默认快捷键



图 3-37　修改快捷键



图 3-38　设置快捷键为 Ctrl+空格

3.7 设置“终端”程序

“终端”程序（terminal）可用于执行 Linux 命令，直接对 Ubuntu 下达命令。安装 Hadoop 时也必须使用“终端”程序。在本书示范介绍过程中会常常用到“终端”程序。为了方便使用，下面先设置“终端”程序的快捷图标。

步骤 01 查找应用程序（见图 3-39）

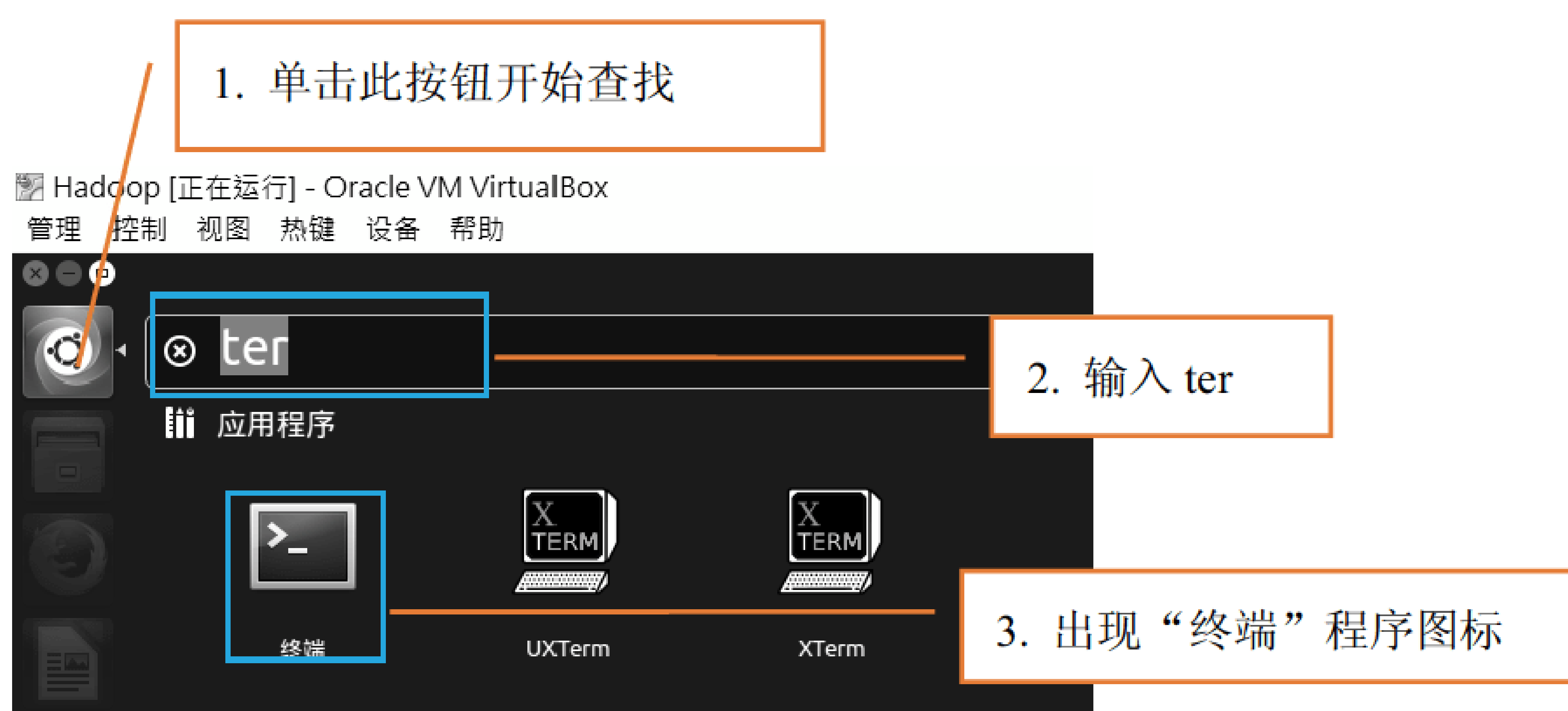


图 3-39 查找应用程序

步骤 02 把“终端”程序图标拖曳到快捷工具栏（见图 3-40）



图 3-40 拖动到快捷工具栏

步骤 03 启动“终端”程序

启动“终端”程序有以下两种方式（见图 3-41）。

（1）单击快捷工具栏上的“终端”程序图标。

（2）使用快捷键 Ctrl + Alt + T。

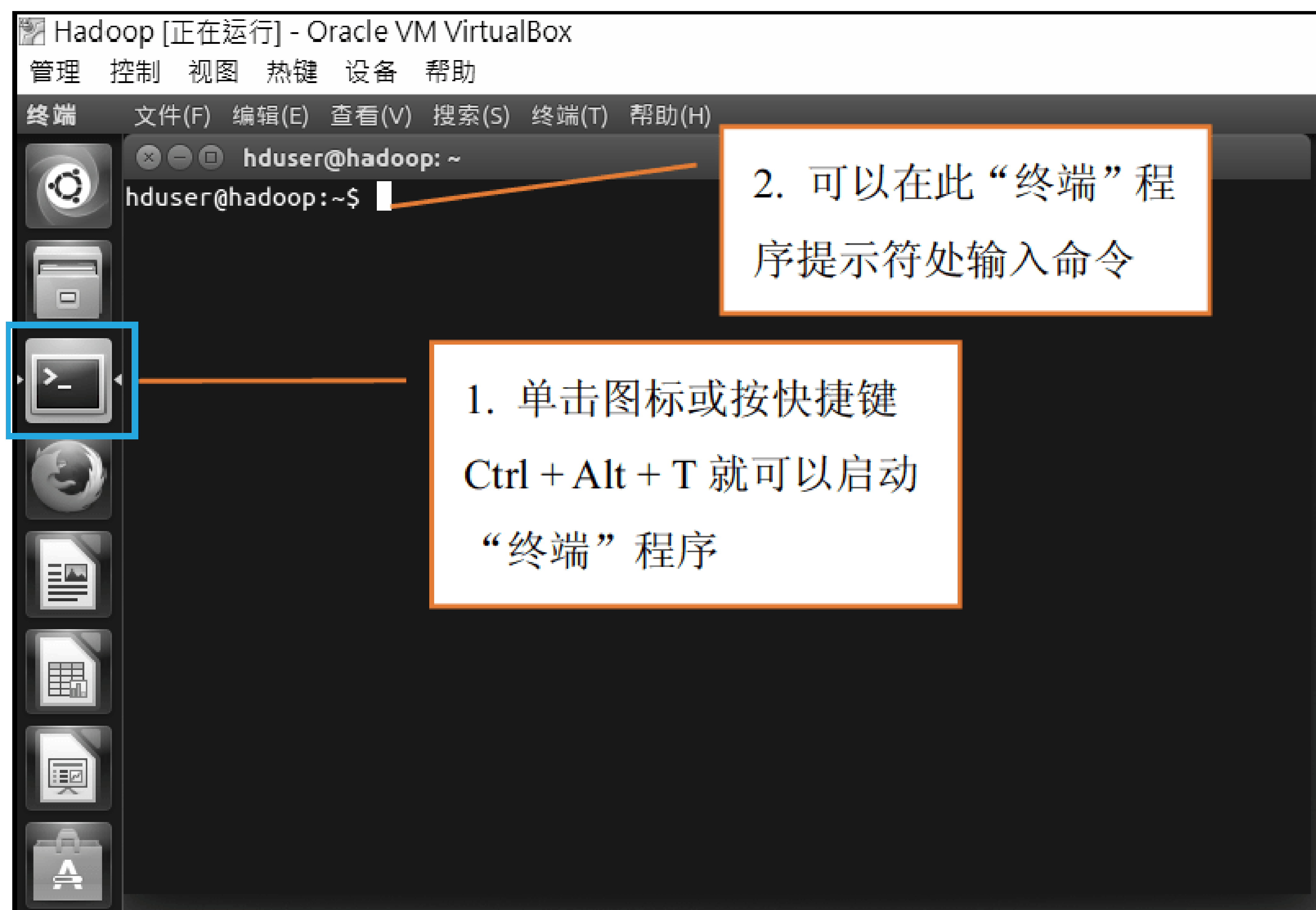


图 3-41 两种启动方法

提示

Ubuntu 默认的“终端”程序提示符格式如下：

[当前用户]@[主机]:[当前目录]$

3.8 设置“终端”程序为白底黑字

用户可以将“终端”程序设置为自己习惯的方式。例如，原本是黑底白字，我们可以将它修改为白底黑字。本书范例中“终端”程序都是以白底黑字显示。

步骤 01 配置文件首选项（见图 3-42）

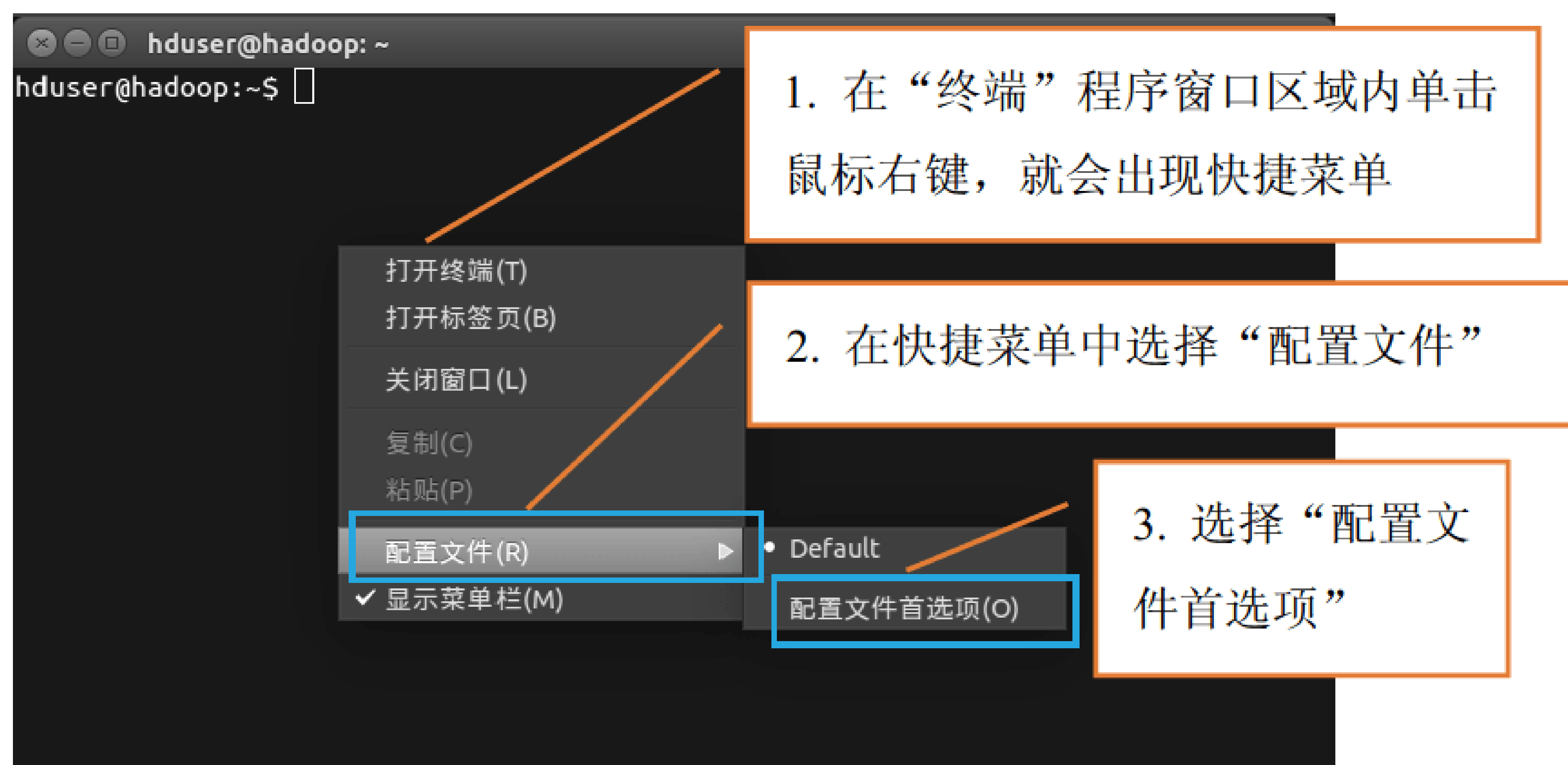


图 3-42 配置文件首选项

步骤 02 选择白底黑字（见图 3-43）

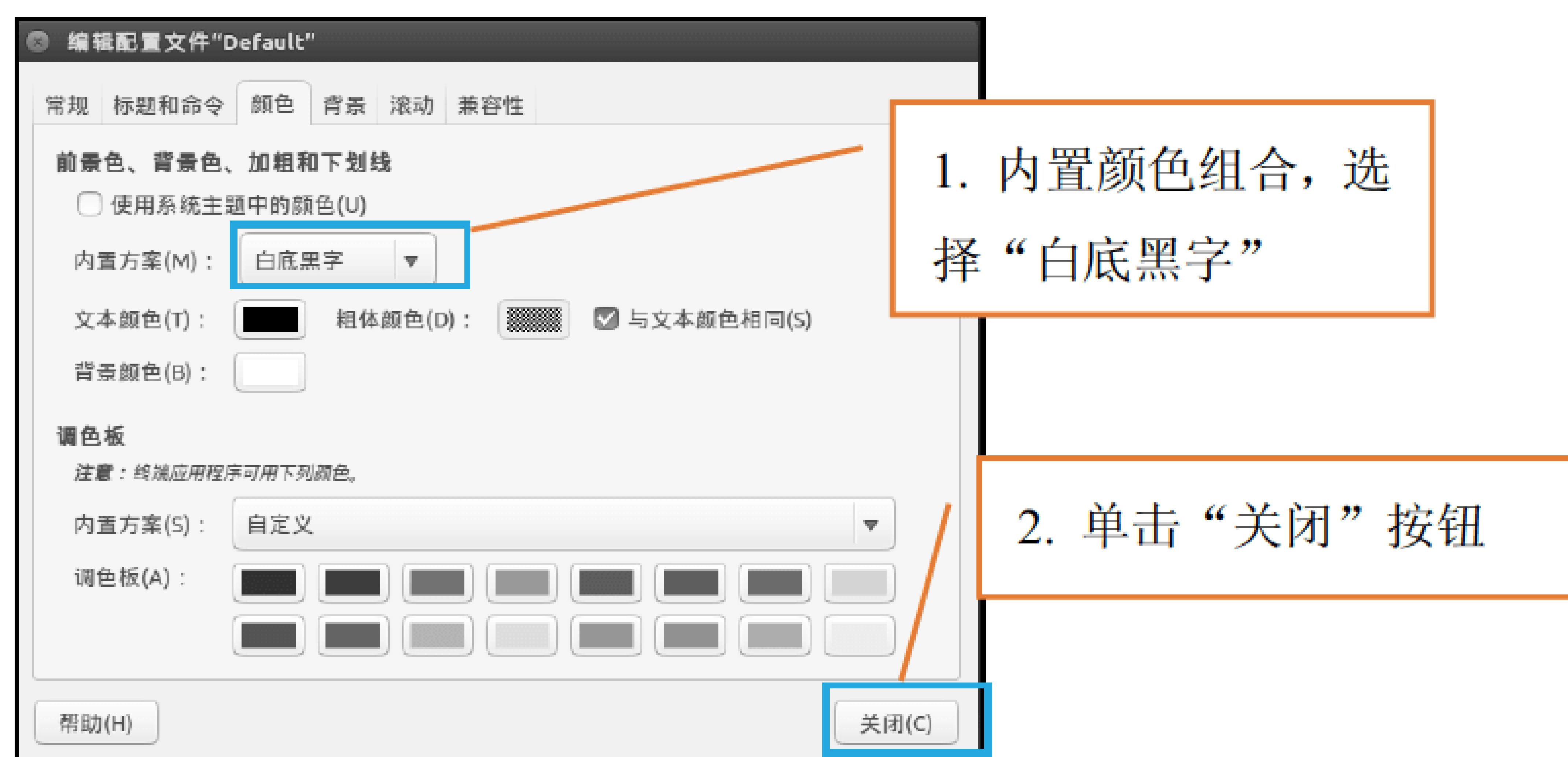


图 3-43　设置内置方案

步骤 03 已设置“终端”程序的颜色组合

设置完成后，“终端”程序的颜色组合为白底黑字，效果 3-44 所示。



图 3-44　显示效果

3.9 设置共享剪贴板

在此说明共享剪贴板功能的方法。例如，原本计算机安装的是 Windows 7 或者 Windows 10，使用 VirtualBox 安装 Ubuntu，则共享剪贴板功能可以让你在 Windows 系统中复制内容，然后粘贴于 Ubuntu。反之亦然，也可以在 Ubuntu 中单击复制，然后在 Windows 中单击粘贴。这个功能很方便，特别是在“终端”程序需要输入命令时，可以使用复制/粘贴节省很多时间，同时也避免了输入错误。注意，在设置共享剪贴板之前，要参照第 3.5 节的说明先安装增强功能（完成 Guest Additions 的安装）。

步骤 01 设置共享剪贴板（见图 3-45）

注：Windows 操作系统中统一都称为“剪贴板”，但是在 Ubuntu 操作系统的中文版中翻译成了“粘贴板”，在本书中，我们还是统一用“剪贴板”，只是在说明 Ubuntu 插图中使用

“粘贴板”。

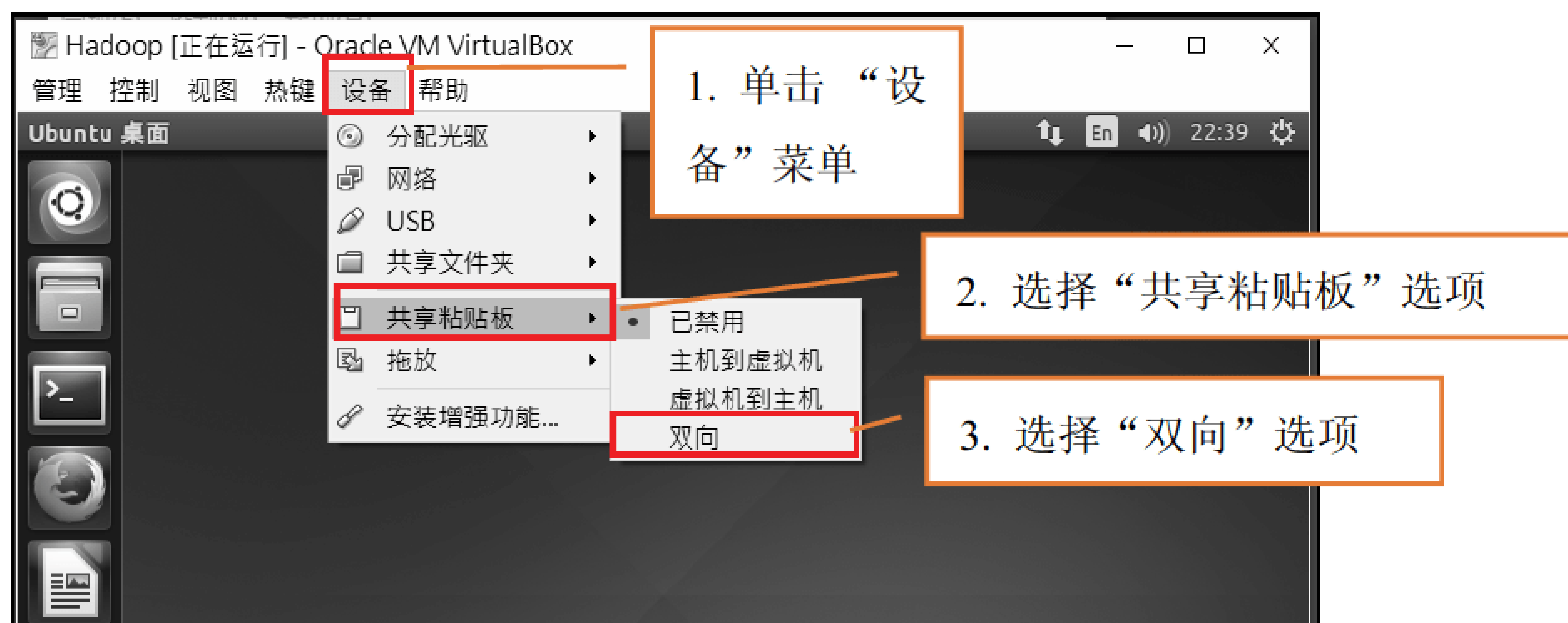


图 3-45　设置共享剪贴板

步骤 02 在 Windows 系统中复制

例如，复制文字“java-version”，如图 3-46 所示。

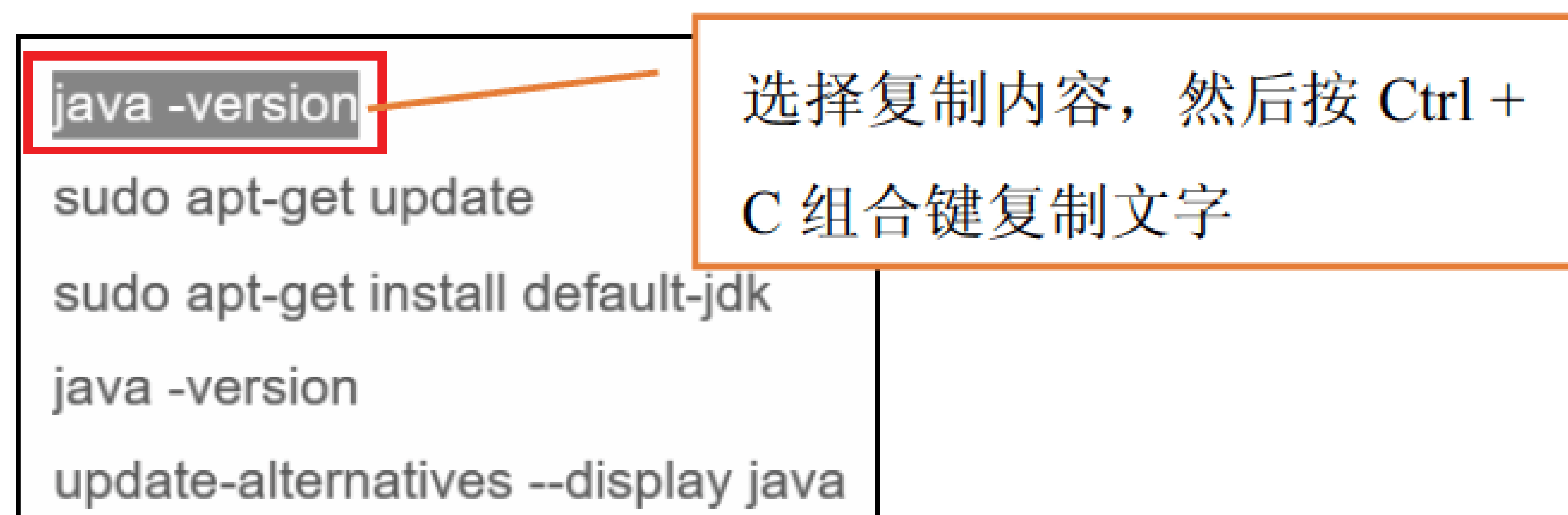


图 3-46　复制文字

步骤 03 在“终端”程序中粘贴

可以使用下列两种方式粘贴之前复制的文字：在“终端”程序中单击鼠标右键，然后单击“粘贴”选项（见图 3-47）；或者按 Ctrl + Shift + V 组合键。



图 3-47　粘贴文字

步骤04 完成粘贴

如图 3-48 所示，已经粘贴了 java -version 命令，按 Enter 键即可开始运行。

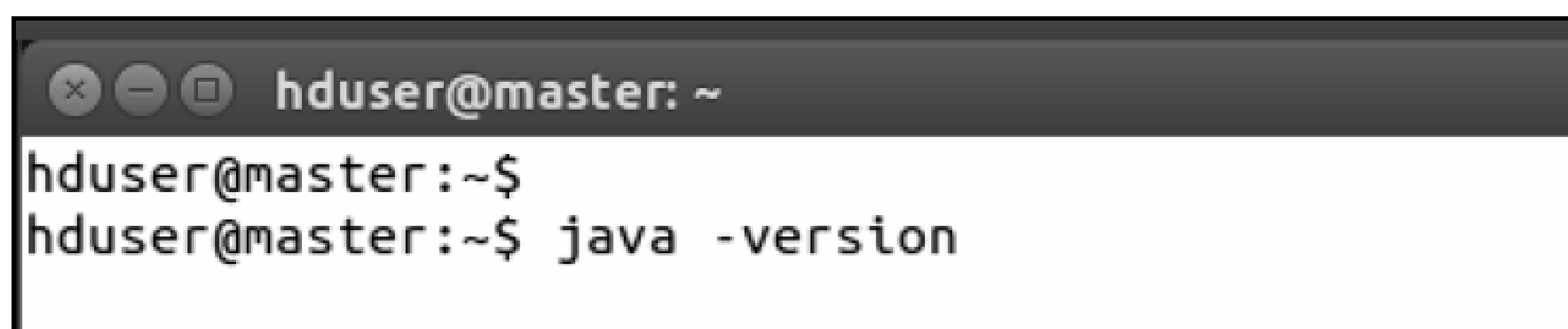


图 3-48　完成粘贴

3.10 设置最佳下载服务器

后续我们会使用 apt-get 安装或更新软件，有时因为下载服务器连接的问题而导致安装或更新出现错误，为了确保能够正确安装或更新软件，请按照下列步骤设置最佳下载服务器。

步骤01 系统设置（见图 3-49）

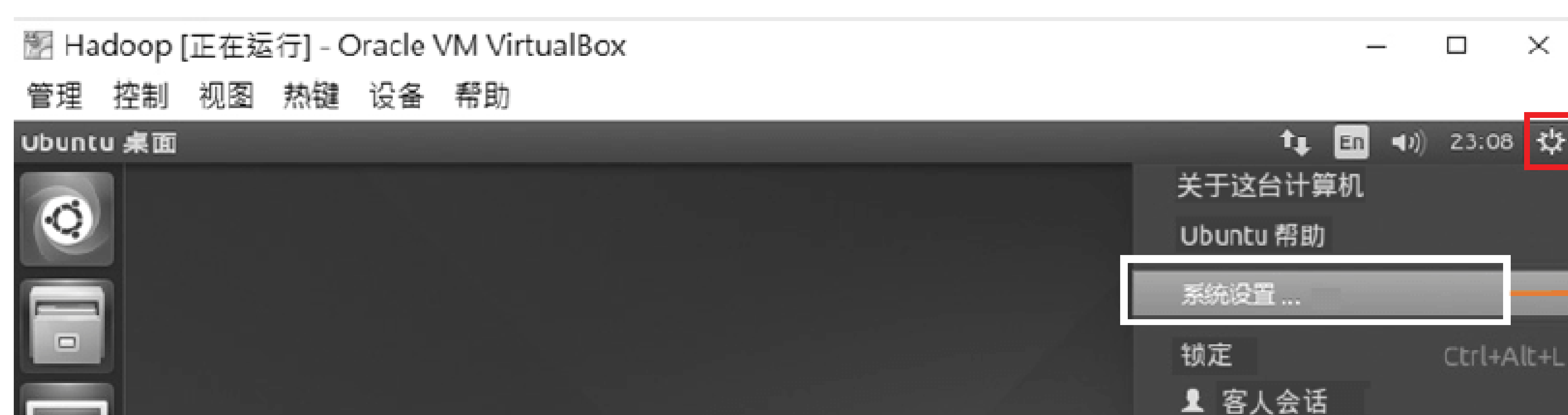


图 3-49　选择“系统设置”

步骤02 单击“软件与更新”（见图 3-50）



图 3-50　单击“软件与更新”

步骤03 单击“下载自”下三角按钮（见图 3-51）



图 3-51　单击下三角按钮

步骤 04 从下拉菜单中选择“中国的服务器”（见图 3-52）

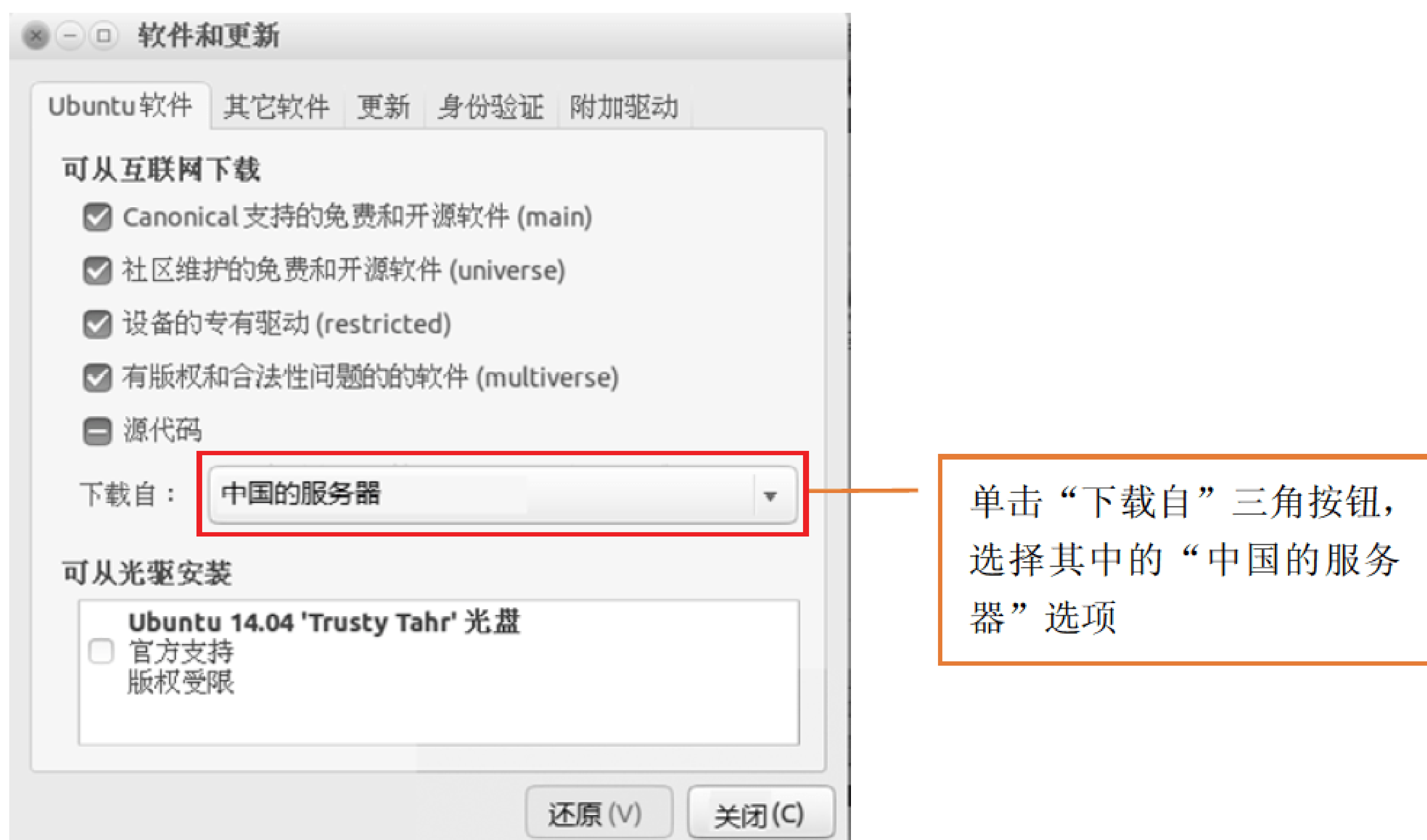


图 3-52　选择“中国的服务器”

步骤 05 选择最佳服务器（见图 3-53）



图 3-53　选择最佳服务器

步骤 06 测试下载服务器（见图 3-54）

系统会经过一连串测试，帮我们找出最佳服务器。



图 3-54　测试下载服务器

步骤 07 选择服务器（见图 3-55）



图 3-55　选择服务器

步骤 08 输入系统管理员密码（见图 3-56）



图 3-56 输入系统管理员密码

步骤 09 单击“关闭”按钮（见图 3-57）



图 3-57 单击“关闭”按钮

步骤 10 单击“重新载入”按钮（见图 3-58）



图 3-58 重新载入

步骤 11 更新缓存

单击“重新载入”会更新缓存，如图 3-59 所示。



图 3-59　更新缓存

步骤 12 完成回到桌面（见图 3-60）

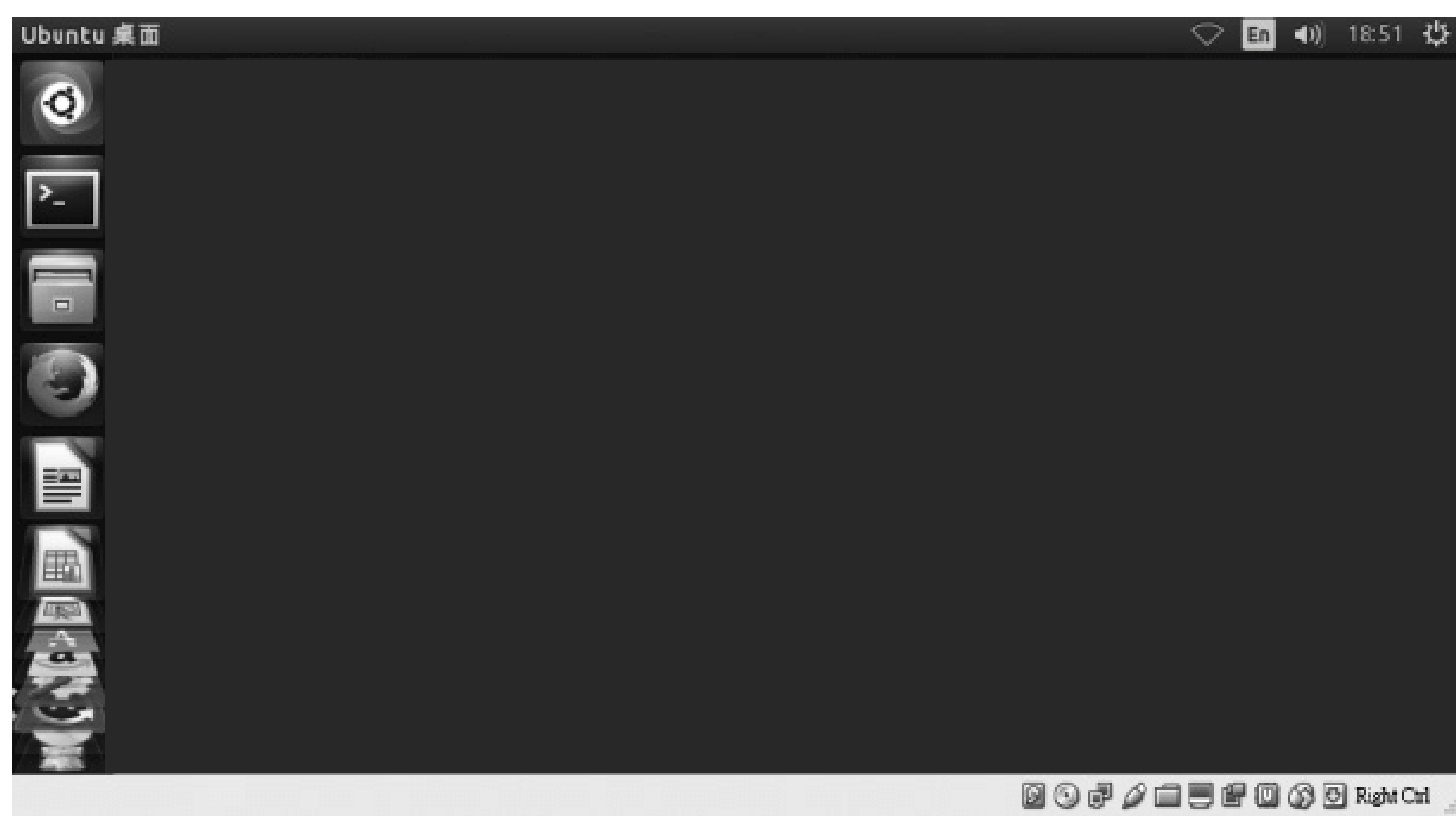


图 3-60　回到桌面

3.11 结论

本章我们介绍了如何安装 Ubuntu 操作系统。下一章，我们将在 Hadoop 虚拟机上安装 Hadoop Single Node Cluster。

第 4 章

Hadoop Single Node Cluster的安装

Hadoop Single Node Cluster 只以一台机器来建立 Hadoop 环境，我们仍然可以使用 Hadoop 命令，只是无法发挥使用多台机器的威力。

Hadoop Single Node Cluster 只有一台服务器，所以所有功能都集中在这台服务器中，如图 4-1 所示。

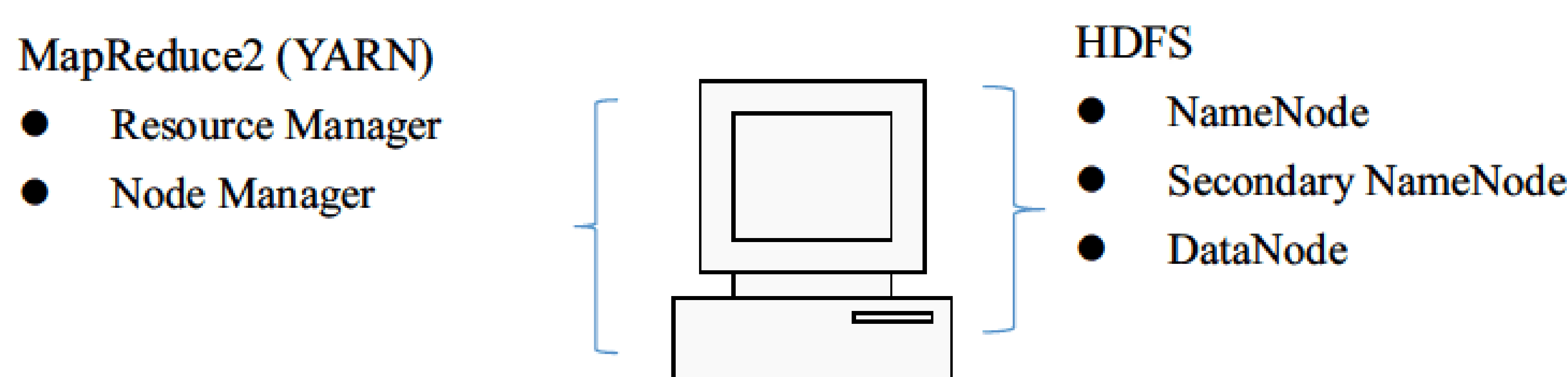


图 4-1　采用单个节点的 Hadoop 集群示意图

安装步骤如表 4-1 所示。

表 4-1　安装步骤

顺序	安装步骤	说明
1	安装 JDK	因为 Hadoop 是使用 Java 开发的，所以必须先安装 JDK
2	设置 SSH 无密码登录	Hadoop 必须通过 SSH 与本地计算机以及其他主机连接，所以必须设置 SSH
3	下载安装 Hadoop	到 Hadoop 官网下载 Hadoop 2.6.4，并安装到 Ubuntu 中
4	设置 Hadoop 环境变量	设置每次用户登录时必须要设置的环境变量
5	Hadoop 配置文件的设置	在 Hadoop 的/usr/local/hadoop/etc/hadoop 目录下有很多配置设置文件，通过编辑这些文件来启用基本或是更高级的功能
6	创建并格式化 HDFS 目录	HDFS 目录是存储 HDFS 文件的地方，在启动 Hadoop 之前必须先创建并格式化 HDFS 目录
7	启动 Hadoop	全部设置完成就可以开始启动 Hadoop，并查看 Hadoop 相关进程是否已经启动
8	打开 Hadoop Web 界面	Hadoop 界面可以让我们查看当前 Hadoop 的状态：Node 节点、应用程序、任务运行状态

➢ **Hadoop 2.6 Single Node Cluster 安装命令**

以下安装过程中所需要输入的命令我们已整理在与本书有关的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可以节省打字的时间，也不用担心输出命令（如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中进行复制和粘贴操作时，请参考第 3.9 节的说明设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

http://blog.sina.com.cn/hadoopsparkbook

4.1 安装 JDK

因为 Hadoop 是以 Java 开发的，所以必须先安装 Java 环境。

步骤 01　启动“终端”程序

接下来，我们将使用“终端”程序下达命令来安装 Hadoop，所以要先启动“终端”程序。我们可以单击快捷工具栏的“终端”程序图标或使用快捷键 Ctrl + Alt + T 启动“终端”程序，如图 4-2 所示。

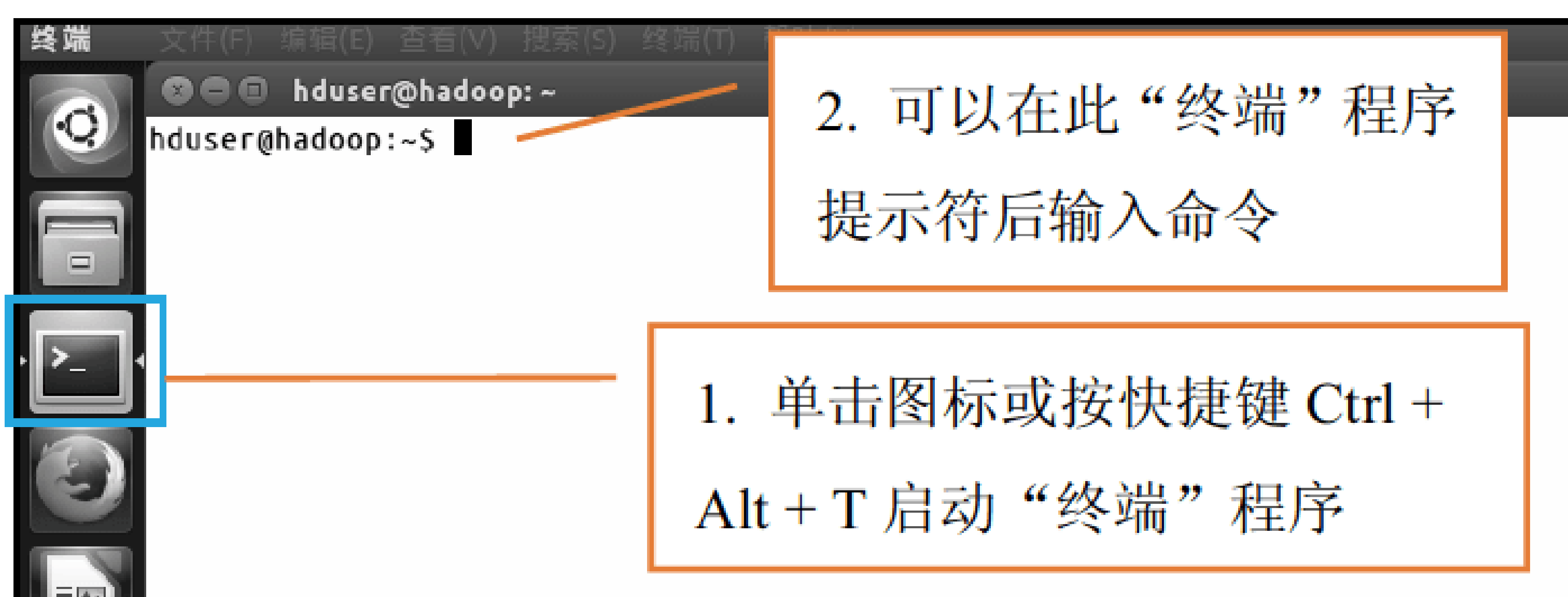


图 4-2　启动“终端”程序

步骤 02　查看当前 Java 版本

在“终端”程序的提示符后输入以下命令并按 Enter 键来运行。

➤ **查看当前 Java 版本**

```
Java -version
```

运行结果如图 4-3 所示。

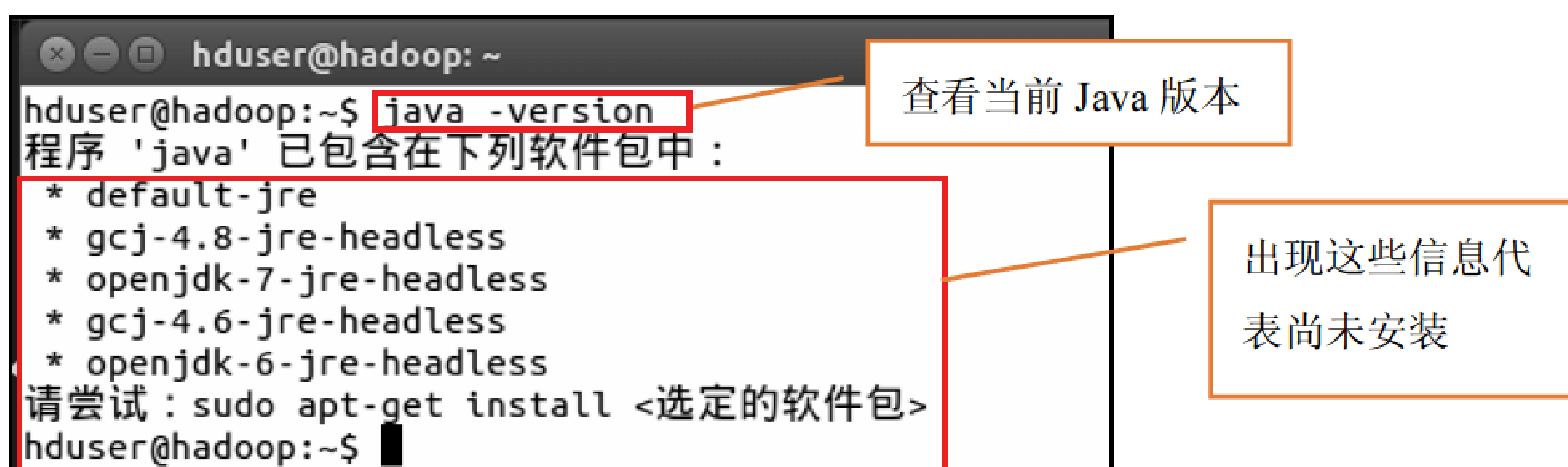


图 4-3　查看当前 Java 版本

步骤 03　sudo apt-get update

在 Linux 中既可使用 apt 进行软件包的管理，也可使用 apt-get 下载安装软件包（或称为套件）。在这里我们会使用 apt-get 安装 jdk。不过，在安装之前，为了获取最新的软件包版本，必须先运行 apt-get update。此命令会连接到 APT Server，更新软件包信息。

运行 apt-get 必须具有 superuser（超级用户）权限，可是 superuser 权限很大，为了安全性考虑，一般我们在运行时不会以 superuser 来登录系统。我们可以在命令前加上 sudo 命令，系统会询问 superuser 密码（安装时输入的密码），这样就可以获得 superuser 权限。可在“终端”程序提示符下输入后面的命令。

➢ 连接到 APT Server，更新软件包信息

```
sudo apt-get update
```

运行后如果显示图4-4所示的界面，系统会让我们输入hduser密码（安装时输入的superuser密码），输入完成后按 Enter 键，系统就会开始获取最新更新的软件包、相关文件的对应列表。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo apt-get update
[sudo] password for hduser:
忽略 http://cn.archive.ubuntu.com trusty InRelease
命中 http://cn.archive.ubuntu.com trusty-updates InRelease
命中 http://cn.archive.ubuntu.com trusty-backports InRelease
命中 http://cn.archive.ubuntu.com trusty Release.gpg
获取：1 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
```

更新软件包信息

图 4-4　更新软件包信息

步骤 04 Install JDK

在“终端”程序提示符下输入以下命令：

➢ 使用 apt-get 安装 JDK

```
sudo apt-get install default-jdk
```

运行后显示如图 4-5 所示的界面，系统会响应：将要占用 39MB 存储空间，询问是否要继续，若要进行，则输入 Y 后按 Enter 键。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo apt-get install default-jdk
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
  ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra
  java-common libatk-wrapper-java libatk-wrapper-java-jni libbonobo2-0
  libbonobo2-common libgconf2-4 libgif4 libgnome2-0 libgnome2-bin
  libgnome2-common libgnomevfs2-0 libgnomevfs2-common libice-dev libidl-common
  libidl0 liborbit-2-0 liborbit2 libpthread-stubs0-dev libsctp1 libsm-dev
  libx11-dev libx11-doc libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
  lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless tzdata
  tzdata-java x11proto-core-dev x11proto-input-dev x11proto-kb-dev
  xorg-sgml-doctools xtrans-dev
建议安装的软件包：
  equivs libbonobo2-bin desktop-base libgnomevfs2-bin libgnomevfs2-extra gamin
  fam gnome-mime-data libice-doc libsm-doc libxcb-doc libxt-doc openjdk-7-demo
  openjdk-7-source visualvm icedtea-7-plugin icedtea-7-jre-jamvm
  sun-java6-fonts fonts-ipafont-gothic fonts-ipafont-mincho ttf-wqy-microhei
  ttf-wqy-zenhei ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts
  ttf-bengali-fonts
下列【新】软件包将被安装：
  ca-certificates-java default-jdk default-jre default-jre-headless
  fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
  libbonobo2-0 libbonobo2-common libgconf2-4 libgif4 libgnome2-0 libgnome2-bin
  libgnome2-common libgnomevfs2-0 libgnomevfs2-common libice-dev libidl-common
  libidl0 liborbit-2-0 liborbit2 libpthread-stubs0-dev libsctp1 libsm-dev
  libx11-dev libx11-doc libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
  lksctp-tools openjdk-7-jdk openjdk-7-jre openjdk-7-jre-headless tzdata-java
  x11proto-core-dev x11proto-input-dev x11proto-kb-dev xorg-sgml-doctools
  xtrans-dev
下列软件包将被升级：
  tzdata
升级了 1 个软件包，新安装了 41 个软件包，要卸载 0 个软件包，有 107 个软件包未被
升级。
需要下载 62.3 MB/62.4 MB 的软件包。
解压缩后会消耗掉 110 MB 的额外空间。
您希望继续执行吗？ [Y/n] Y
```

1. 安装 JDK

2. 输入 Y 再按 Enter 键

图 4-5　安装 JDK

步骤 05　再次查询 Java 版本

再一次在“终端”程序的提示符下输入以下命令：

➤ **查询 Java 版本**

```
Java -version
```

系统响应已安装的 Java 版本时代表已经成功安装了 JDK，如图 4-6 所示。

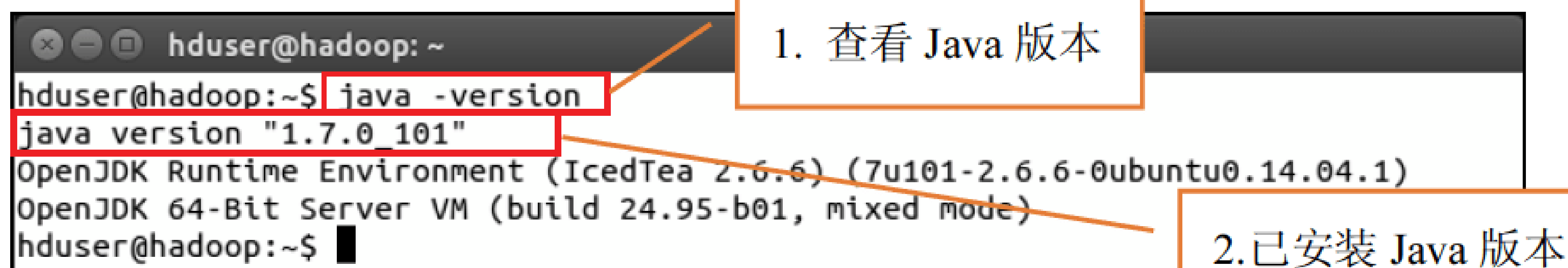


图 4-6　查询 Java 版本

步骤 06　查询 Java 安装的位置

想知道 Java 安装在哪里时，可以使用下列命令：

➤ **查询 Java 安装路径**

```
update-alternatives --display Java
```

系统会响应安装的路径，如图 4-7 所示。

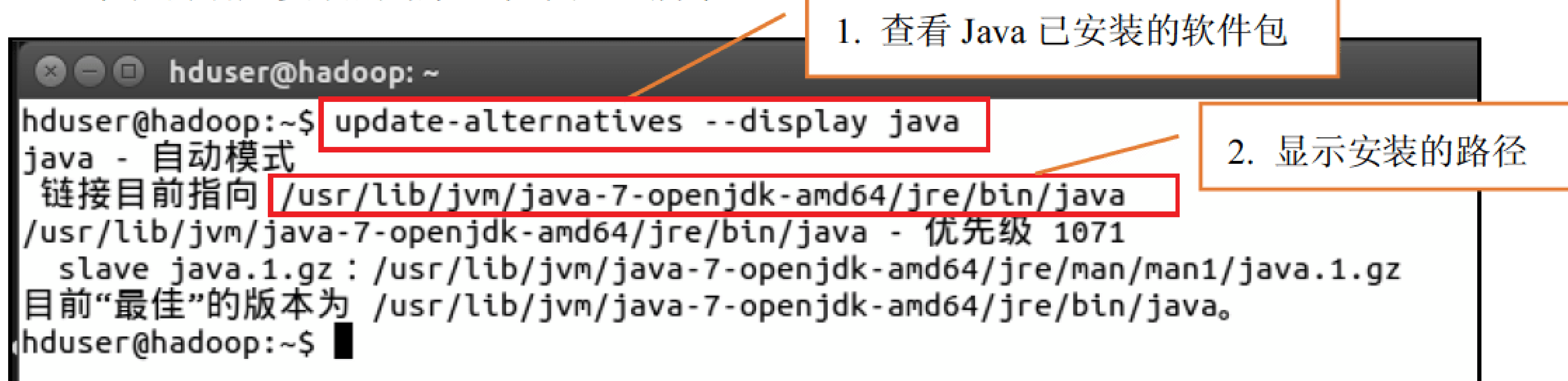


图 4-7　查询 Java 安装路径

从图 4-7 可以得知，Java 安装在 /usr/lib/jvm/java-7-openjdk-amd64 文件夹中，稍后我们会在~/.bashrc 文件中设置此路径。

4.2 设置 SSH 无密码登录

Hadoop 是由很多台服务器所组成的，当我们启动 Hadoop 系统时，NameNode 必须与 DataNode 连接并管理这些节点（DataNode）。此时系统会要求用户输入密码。为了让系统顺利运行而不手动输入密码，需要将 SSH 设置成无密码登录。注意，无密码登录并非不需要密码，而是以事先交换的 SSH Key（密钥）来进行身份验证。

如图 4-8 所示，Hadoop 使用 SSH（Secure Shell）连接，这是目前较可靠、专为远程登录

其他服务器提供的安全性协议。通过 SSH 会对所有传输的数据进行加密，利用 SSH 协议可以防止远程管理系统时信息外泄的问题。

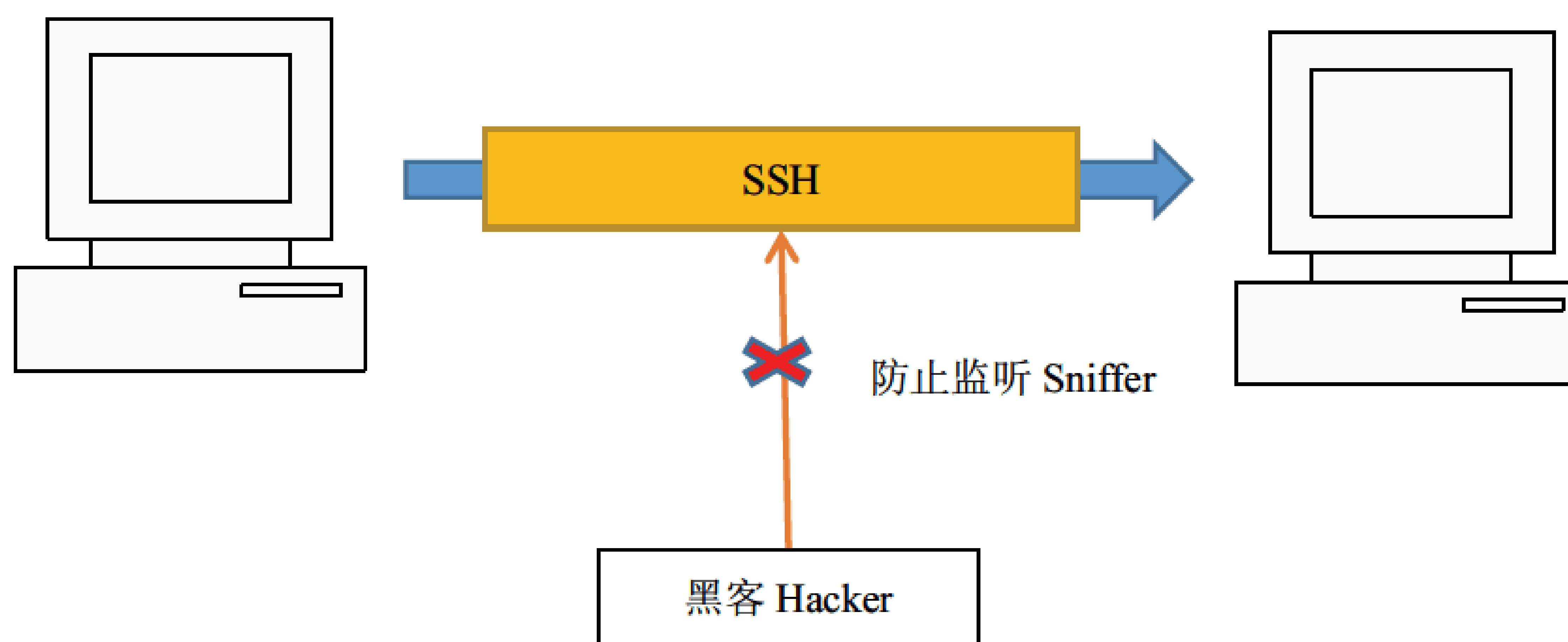


图 4-8　Hadoop 使用 SSH 连接

步骤 01 安装 SSH

在“终端”程序提示符下输入以下命令：

➢ **安装 SSH**

```
sudo apt-get install ssh
```

运行后显示如图 4-9 所示的界面，系统会询问是否要继续，若继续，请输入“Y”再按 Enter 键。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo apt-get install ssh
[sudo] password for hduser:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
  libck-connector0 ncurses-term openssh-client openssh-server
  openssh-sftp-server ssh-import-id
建议安装的软件包：
  libpam-ssh keychain monkeysphere rssh molly-guard
下列【新】软件包将被安装：
  libck-connector0 ncurses-term openssh-server openssh-sftp-server ssh
  ssh-import-id
下列软件包将被升级：
  openssh-client
升级了 1 个软件包，新安装了 6 个软件包，要卸载 0 个软件包，有 106 个软件包未被升
级。
需要下载 620 kB/1,183 kB 的软件包。
解压缩后会消耗掉 3,450 kB 的额外空间。
您希望继续执行吗？ [Y/n]
```

1. 输入 sudo apt-get install ssh

2. 输入 Y 再按 Enter 键

图 4-9　安装 SSH

步骤 02 安装 rsync

在“终端”程序提示符下输入以下命令：

➢ **安装 rsync**

```
sudo apt-get install rsync
```

运行后界面如图 4-10 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo apt-get install rsync
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
rsync 已经是最新的版本了。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 252 个软件包未被升级。
hduser@hadoop:~$

升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 157 个软件包未被升级。
需要下载 1,183 KB 的软件包文件。
此操作完成之后，会多占用 3,450 KB 的磁盘空间。
Do you want to continue? [Y/n]
```

1. 使用 apt-get 安装 rsync

2. 输入 Y

图 4-10　安装 rsync

步骤 03　产生 SSH Key（密钥）

在“终端”程序提示符下输入以下命令：

➢ **产生 SSH Key（密钥）进行后续身份验证**

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

运行后界面如图 4-11 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
Generating public/private dsa key pair.
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_dsa.
Your public key has been saved in /home/hduser/.ssh/id_dsa.pub.
The key fingerprint is:
c1:e3:7a:e0:17:5a:73:89:63:02:fe:fc:b8:3c:1f:05 hduser@hadoop
The key's randomart image is:
+--[ DSA 1024]----+
|                 |
|        .        |
|     .   E       |
|    . . . = .    |
|     . o S +     |
|      + B *      |
|       * +       |
|      ..= .      |
|       ++o       |
+-----------------+
```

1. 使用 ssh-keygen 产生密钥

2. 产生的密钥文件

图 4-11　产生 SSH Key

步骤 04　查看产生的 SSH Key（密钥）

SSH Key（密钥）会产生在用户的根目录下，也就是/home/hduser。

➢ **查看产生的 SSH Key（密钥）**

```
ll ~/.ssh
```

运行后界面如图 4-12 所示。

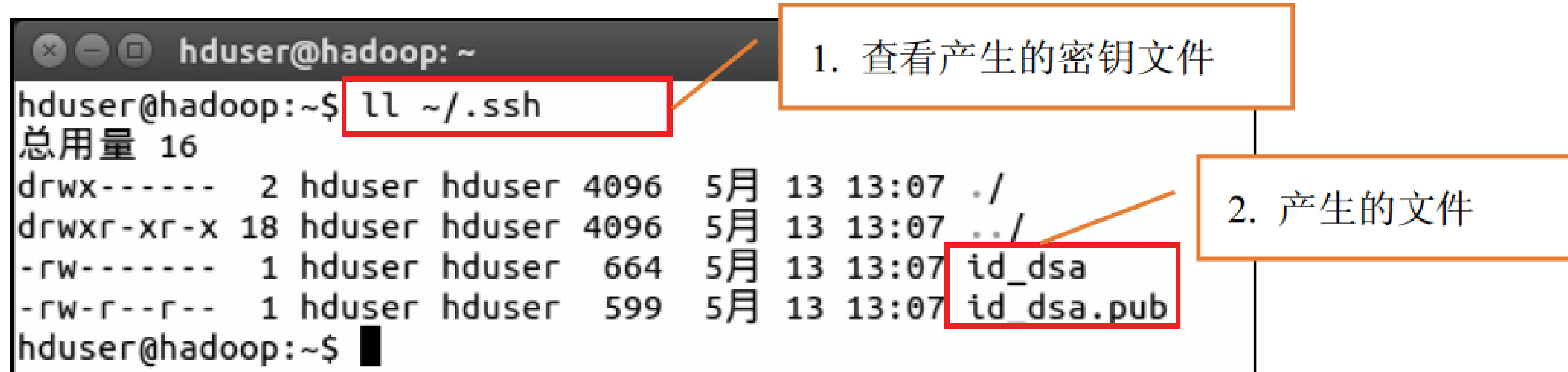


图 4-12　查看产生的 SSH Key

步骤 05 将产生的 Key 放置到许可证文件中

为了能够无密码登录本机，我们必须将产生的公钥加入许可证文件中。
在终端程序提示符下输入以下命令：

➢ **将产生的 Key 放置到许可证文件中**

```
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

命令运行后，会将~/.ssh/id_dsa.pub 附加到~/.ssh/authorized_keys 许可证文件之后。



Linux 的输出重定向附加功能命令的格式如下：

```
命令 >> 文件
```

这个重定向符号“>>”会将命令运行后产生的标准输出（stdout）重定向附加在该文件之后。
（1）如果文件不存在，就会先创建一个新文件，然后把标准输出（stdout）的内容存储在这个文件中。
（2）如果文件已经存在，就会将标准输出（stdout）的数据附加至文件内容的后面，而不会覆盖原来文件的内容。

运行后显示界面如图 4-13 所示。

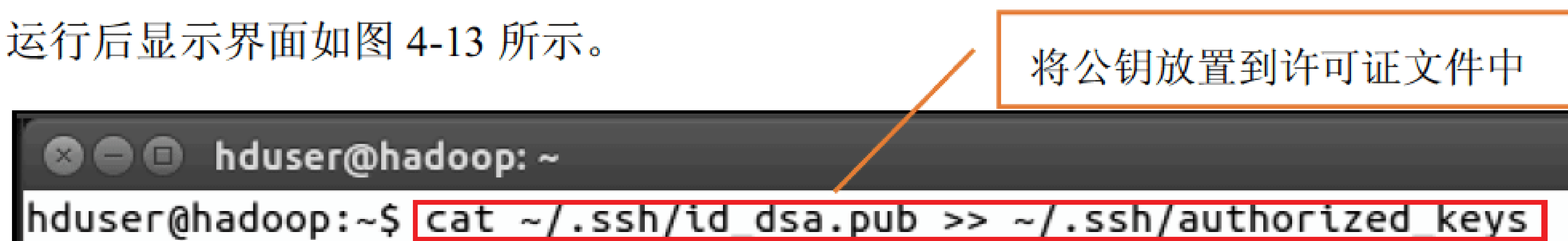


图 4-13　将产生的 key 放置到许可证文件中

4.3 下载安装 Hadoop

接下来到 Hadoop 官网下载 Hadoop 2.6.4 版本，并且安装到 Ubuntu 中。请注意，在这里

一定要下载 Hadoop 2.6.4，因为后续我们要安装 Spark 2.0 版，与 Spark 2.0 配合的 Hadoop 版本我们选择 2.6.4。

步骤 01 连接至 Hadoop 下载页面（见图 4-14）

在浏览器输入下列网址：

➤ **连接至 Hadoop 官网下载页面**

```
https://archive.apache.org/dist/hadoop/common/
```

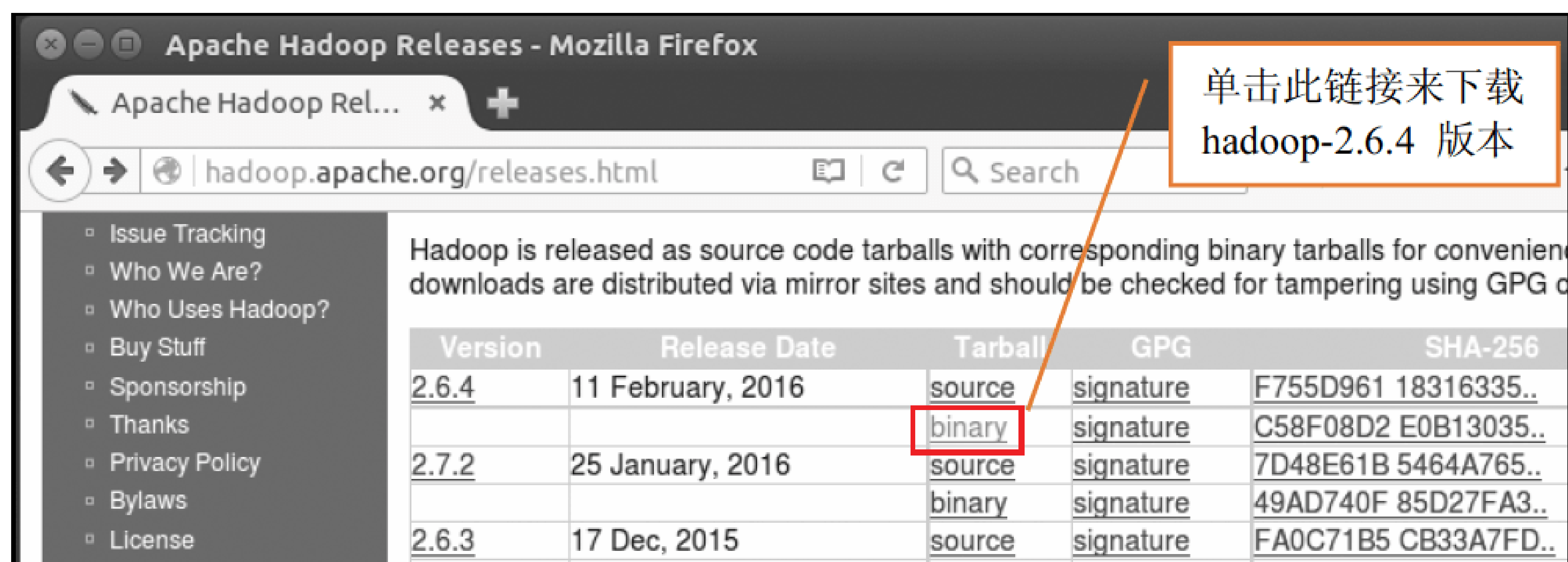


图 4-14　连接至 Hadoop 下载页面

步骤 02 复制链接（见图 4-15）

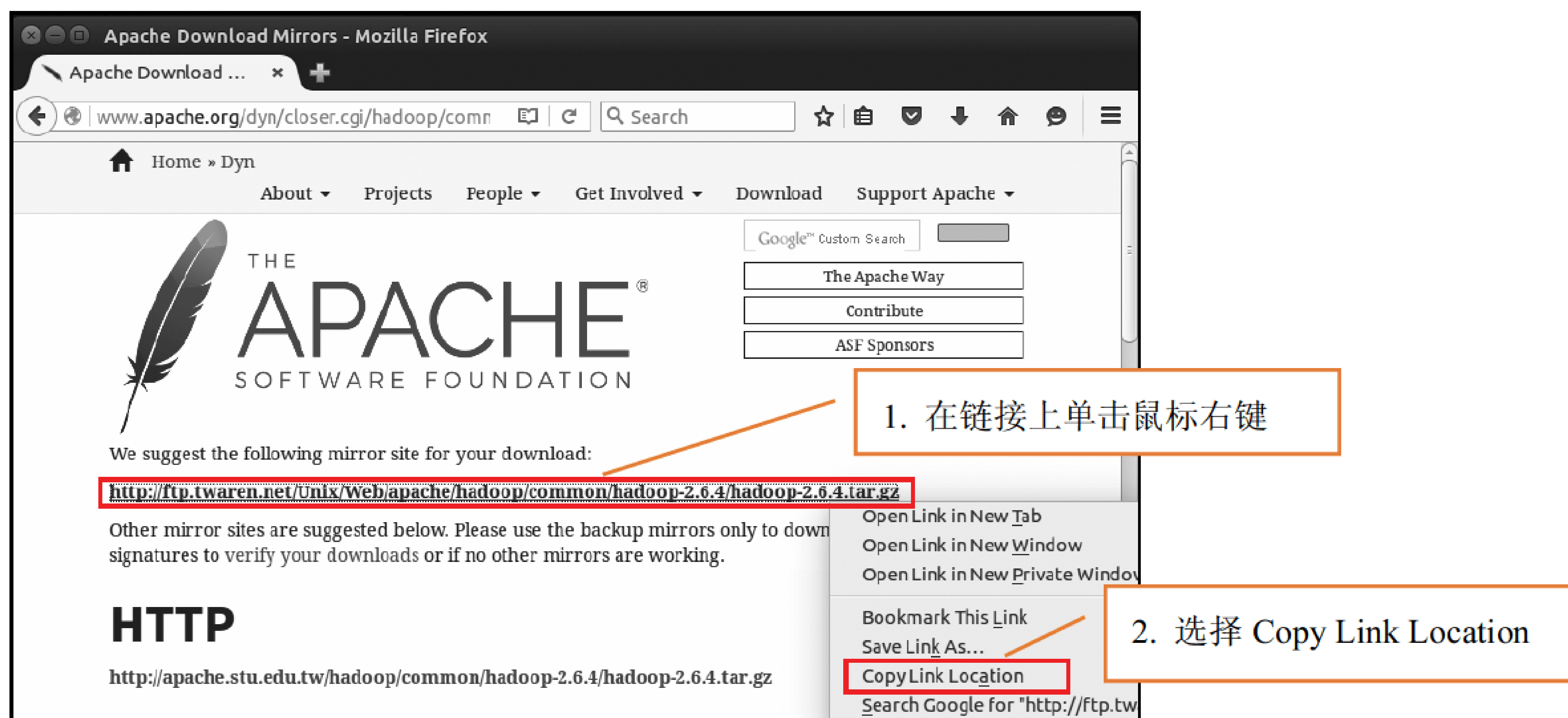


图 4-15　复制链接

步骤 03 执行下载命令

在“终端”程序提示符下输入 wget 及空格键，然后按 Ctrl + Shift + V 组合键粘贴之前复制的链接，如下列命令：

➢ **下载 Hadoop-2.6.4.tar.gz**

```
Wget https://archive.apache.org/dist/hadoop/common/hadoop-2.6.4/hadoop-2.6.4.tar.gz
```

执行结果如图 4-16 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ wget http://apache.stu.edu.tw/hadoop/common/hadoop-2.6.4/hadoop
-2.6.4.tar.gz
--2016-08-22 11:35:01--  http://apache.stu.edu.tw/hadoop/common/hadoop-2.6.4/had
oop-2.6.4.tar.gz
正在解析主机 apache.stu.edu.tw (apache.stu.edu.tw)... 120.119.118.1, 2001:e10:c4
1:eeee::1
正在连接 apache.stu.edu.tw (apache.stu.edu.tw)|120.119.118.1|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 196015975 (187M) [application/x-gzip]
Saving to: 'hadoop-2.6.4.tar.gz'

100%[======================================>] 196,015,975 2.18MB/s   in 97s

2016-08-22 11:36:39 (1.92 MB/s) - 'hadoop-2.6.4.tar.gz' saved [196015975/1960159
75]
```

图 4-16　执行下载命令

步骤 04 解压缩 hadoop 2.6.4

在“终端”程序提示符下输入以下命令：

➢ **解压缩 hadoop-2.6.4.tar.gz 至 hadoop-2.6.4 目录**

```
sudo tar -zxvf Hadoop-2.6.4.tar.gz
```

执行结果如图 4-17 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo tar -zxvf hadoop-2.6.4.tar.gz
```

图 4-17　解压缩 hadoop 2.6.4

步骤 05 将 Hadoop 移动到/usr/local

软件默认的安装路径是/usr/local，可在“终端”程序提示符下输入以下命令：

➢ **移动 hadoop-2.6.4 目录到/usr/local/hadoop**

```
sudo mv hadoop-2.6.4 /usr/local/hadoop
```

运行后界面如图 4-18 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo mv hadoop-2.6.4 /usr/local/hadoop
```

图 4-18　将 Hadoop 移动到 /usr/local

步骤 06 查看 Hadoop 安装目录/usr/local/hadoop

在"终端"程序中输入下列命令：

➢ **查看 Hadoop 安装目录/usr/local/hadoop**

```
ll /usr/local/hadoop
```

运行后界面如图 4-19 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ ll /usr/local/hadoop
总用量 60
drwxr-xr-x  9 10011 10011  4096 1月 26  2016 ./
drwxr-xr-x 11 root  root   4096 8月  2 12:52 ../
drwxr-xr-x  2 10011 10011  4096 1月 26  2016 bin/
drwxr-xr-x  3 10011 10011  4096 1月 26  2016 etc/
drwxr-xr-x  2 10011 10011  4096 1月 26  2016 include/
drwxr-xr-x  3 10011 10011  4096 1月 26  2016 lib/
drwxr-xr-x  2 10011 10011  4096 1月 26  2016 libexec/
-rw-r--r--  1 10011 10011 15429 1月 26  2016 LICENSE.txt
-rw-r--r--  1 10011 10011   101 1月 26  2016 NOTICE.txt
-rw-r--r--  1 10011 10011  1366 1月 26  2016 README.txt
drwxr-xr-x  2 10011 10011  4096 1月 26  2016 sbin/
drwxr-xr-x  4 10011 10011  4096 1月 26  2016 share/
hduser@hadoop:~$
```

图 4-19 查看 Hadoop 安装目录/usr/local/hadoop

常用的目录说明如表 4-2 所示。

表 4-2 常用的目录说明

目录	说明
bin/	各项运行文件，包括 Hadoop、HDFS、YARN 等
sbin/	各项 shell 运行文件，包括 start-all.sh、stop-all.sh
etc/	etc/hadoop 子目录包含 Hadoop 配置文件，例如 hadoop-env.sh、core-site.xml、YARN-site.xml、mapred-site.xml、hdfs-site.xml
lib/	Hadoop 函数库
logs/	系统日志，可以查看系统运行状况，运行有问题时可从日志找出错误原因

4.4 设置 Hadoop 环境变量

运行 Hadoop 必须设置很多环境变量，可是如果每次登录时都必须重新设置就会很麻烦，因此我们可以在~/.bashrc 文件中设置每次登录时都会自动运行一次环境变量设置。

步骤 01 编辑~/.bashrc

在"终端"程序中输入下列命令：

➢ **编辑~/.bashrc**

```
sudo gedit ~/.bashrc
```

输入后按 Enter 键就会打开~/.bashrc，如图 4-20 所示。

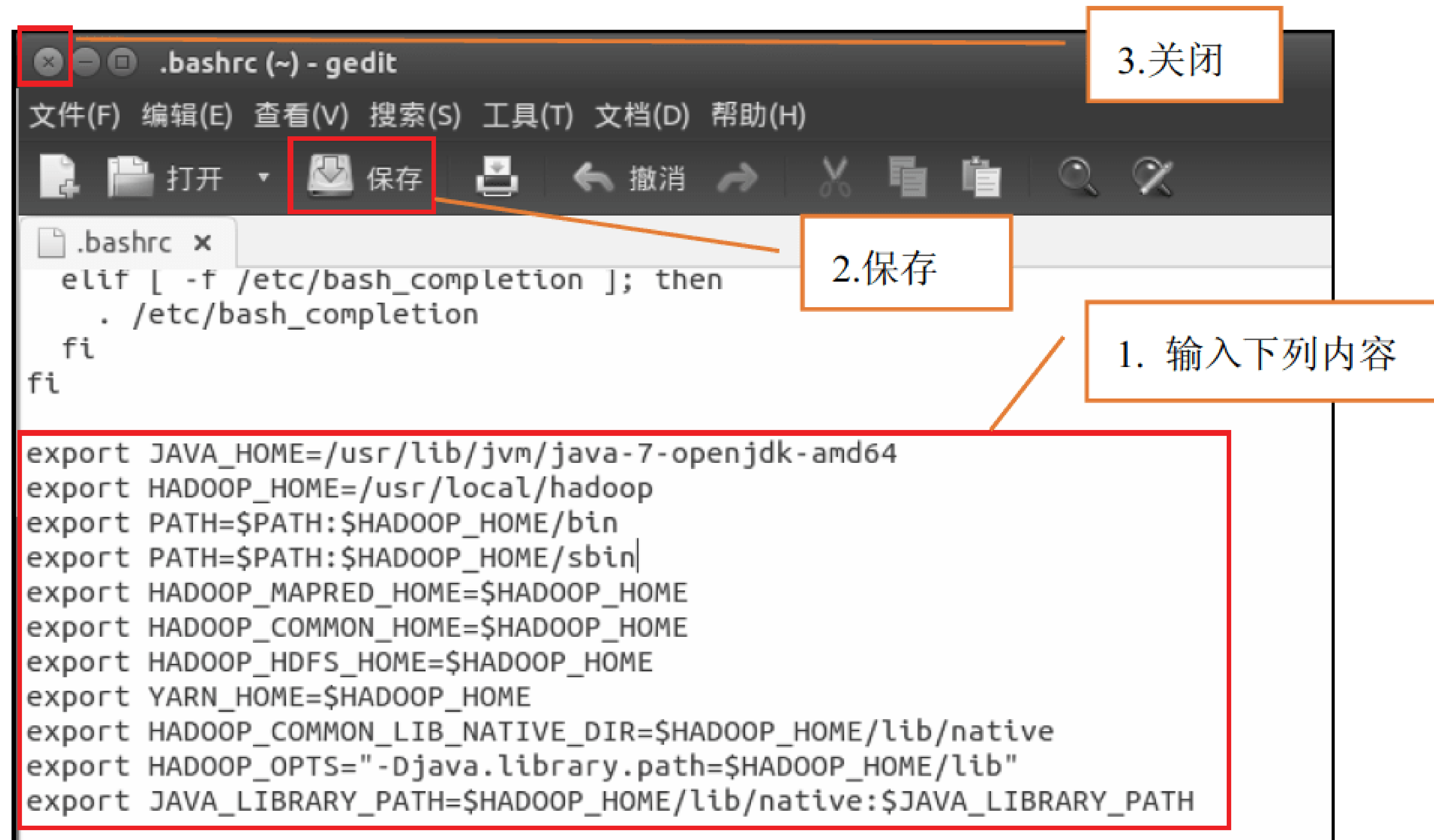


图 4-20　编辑~/.bashrc

编辑完成后，先保存，再关闭 gedit。

上述设置说明如下：

➢ **设置 JDK 安装路径（参考第 4.2 节）**

```
export JAVA_HOME=/usr/lib/jvm/Java-7-openjdk-amd64
```

➢ **设置 HADOOP_HOME 为 Hadoop 的安装路径/usr/local/hadoop**

```
export HADOOP_HOME=/usr/local/hadoop
```

➢ **设置 PATH**

```
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
```

Hadoop 运行文件目录是/bin 与/sbin。设置 PATH 可以让你在其他目录时仍然能够运行 Hadoop 命令。

➢ **设置 HADOOP 其他环境变量**

设置这些环境变量为$HADOOP_HOME，也就是 Hadoop 的安装路径/usr/local/hadoop。

```
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
```

```
export YARN_HOME=$HADOOP_HOME
```

➤ **链接库的相关设置**

```
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-DJava.library.path=$HADOOP_HOME/lib"
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
```

步骤 02 让 ~/.bashrc 修改的设置生效

当我们修改~/.bashrc 之后，先从系统注销再登录系统，这样设置就会生效，或者使用 source 命令让~/.bashrc 设置生效。请在“终端”程序中输入下列命令：

➤ **让~/.bashrc 设置生效**

```
source ~/.bashrc
```

运行结果如图 4-21 所示。

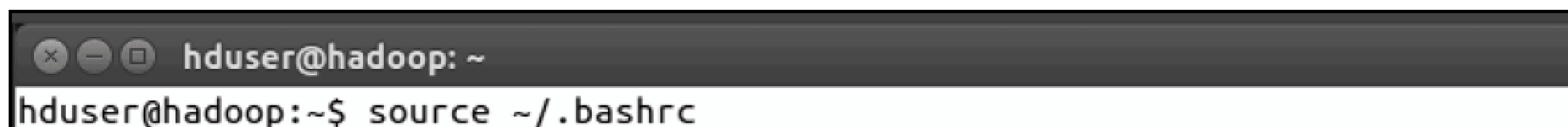


图 4-21　让~/.bashrc 修改的设置生效

4.5 修改 Hadoop 配置设置文件

接下来要进行 Hadoop 配置设置，包括 Hadoop-env.sh、core-site.xml、YARN-site.xml、mapred-site.xml、hdfs-site.xml。

步骤 01 设置 hadoop-env.sh 配置文件

hadoop-env.sh 是 Hadoop 的配置文件，在这里必须设置 Java 的安装路径，在“终端”程序中输入下列命令：

➤ **编辑 Hadoop-env.sh**

```
sudo gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

输入后按 Enter 键就会打开 hadoop-env.sh，屏幕显示界面如图 4-22 所示。

原本文件中 JAVA_HOME 的设置是：

```
export JAVA_HOME=${JAVA_HOME}
```

修改为：

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

编辑完成后，先保存，再关闭 gedit。

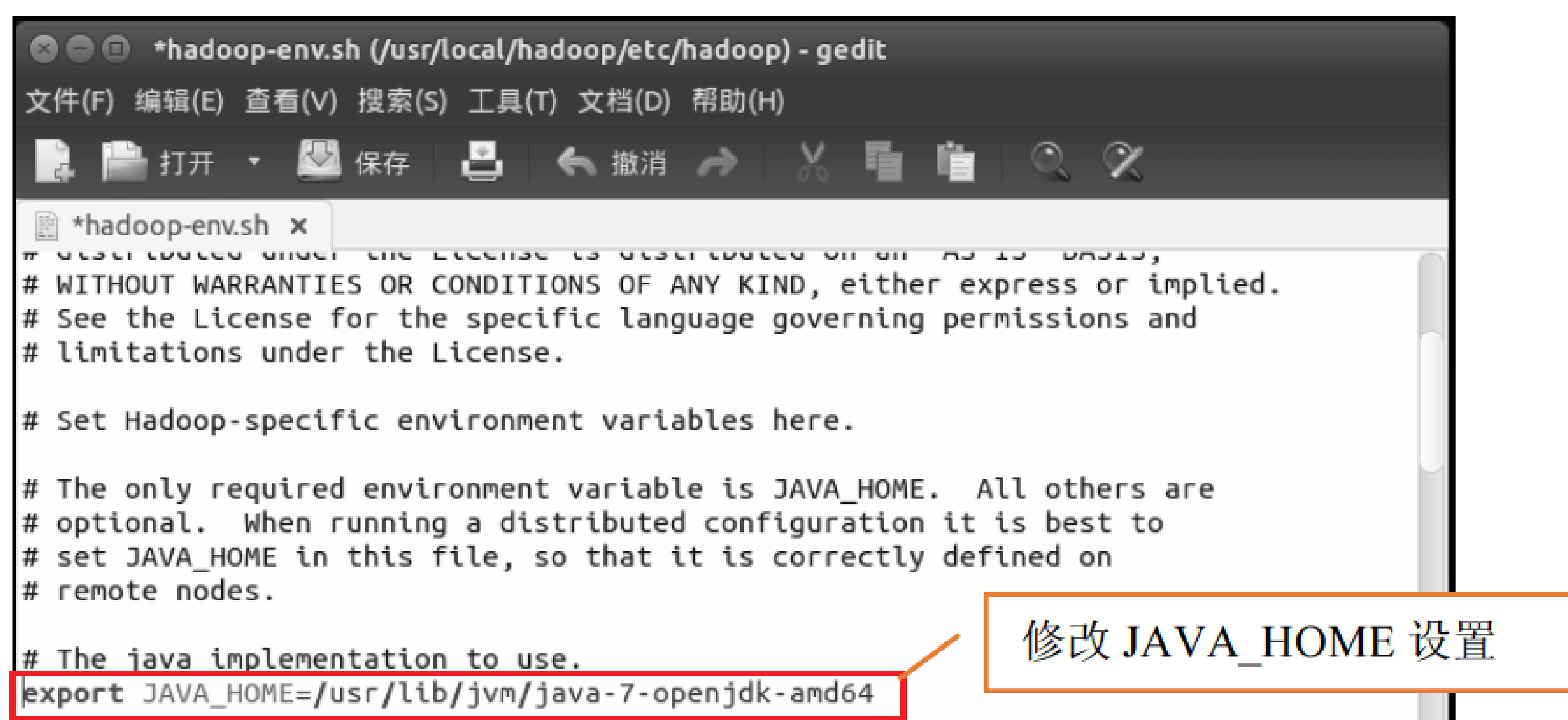


图 4-22　设置 hadoop-env.sh 配置文件

步骤 02 设置 core-site.xml

在“终端”程序输入下列命令：

➢ **修改** core-site.xml

```
sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml
```

输入后按 Enter 键就会打开 core-site.xml，屏幕显示界面如图 4-23 所示。

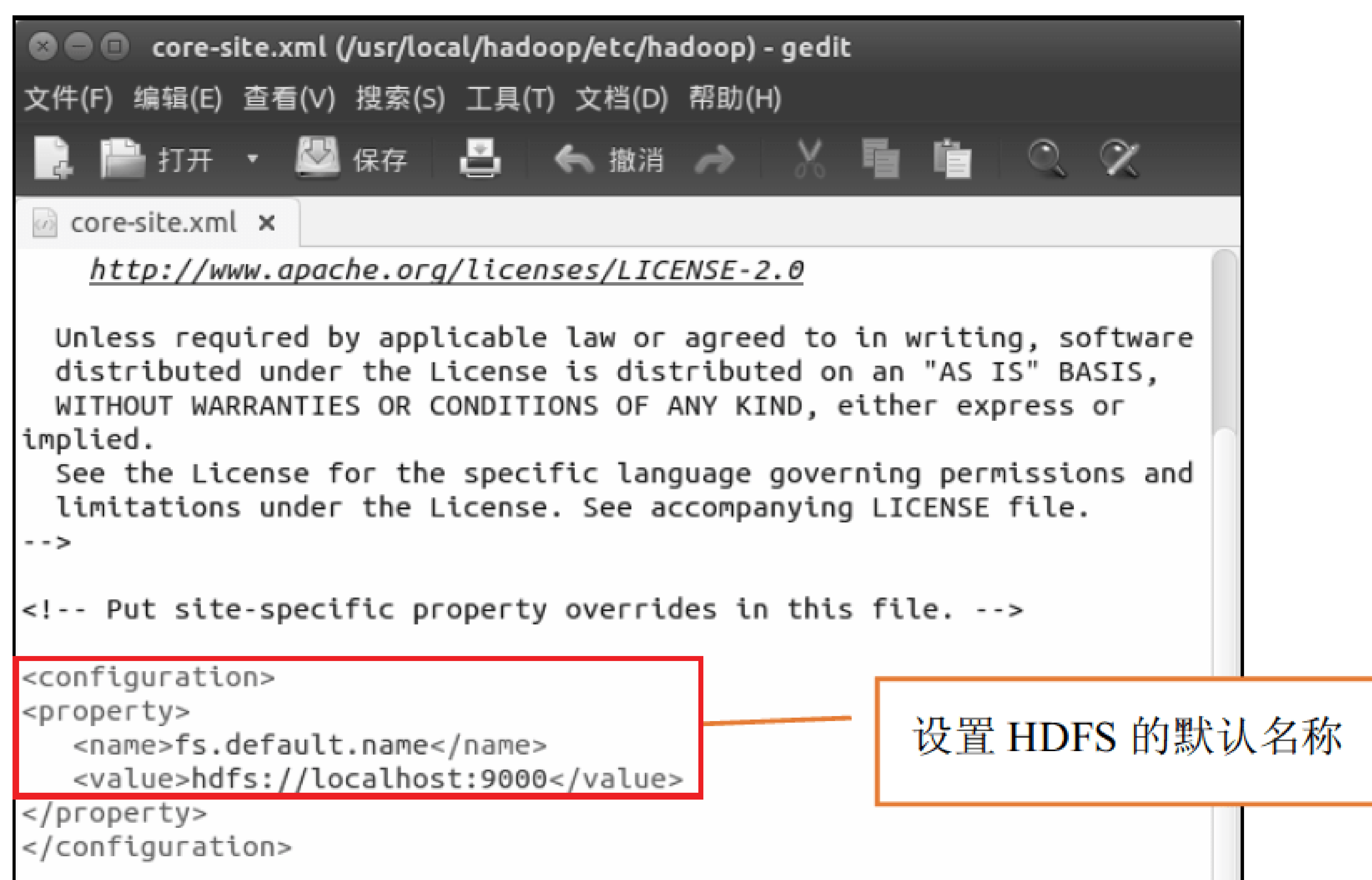


图 4-23　设置 core-site.xml

在 core-site.xml 中，我们必须设置 HDFS 的默认名称，当我们使用命令或程序要存取 HDFS 时，可使用此名称。编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 YARN-site.xml

YARN-site.xml 文件中含有 MapReduce2（YARN）相关的配置设置，可在“终端”程序输

入下列命令：

> **编辑 YARN-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

输入后按 Enter 键就会打开 YARN-site.xml，请在 <configuration></configuration> 之间输入图 4-24 中显示的内容。

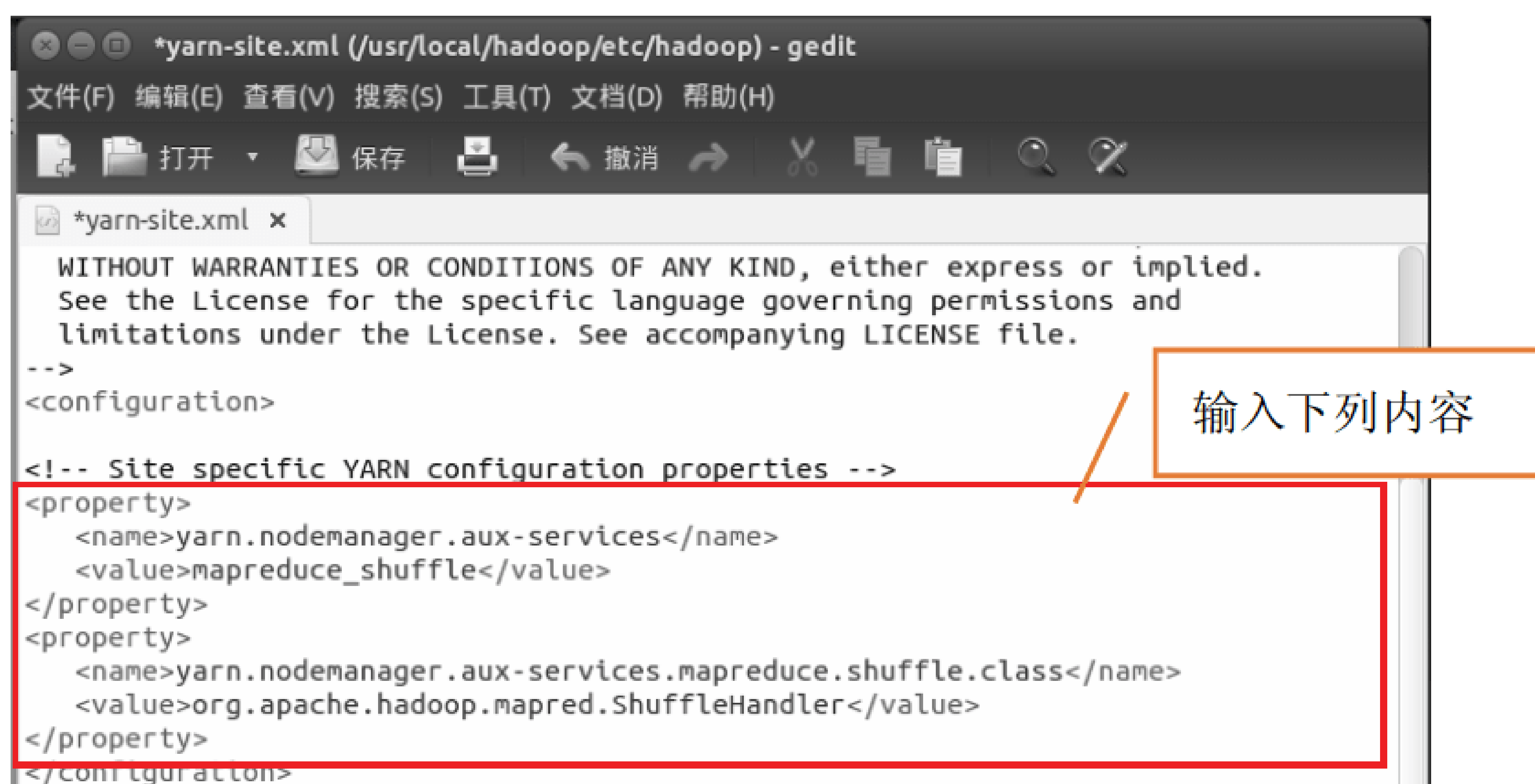


图 4-24　输入新内容

编辑完成后，先保存，再关闭 gedit。

步骤 04　设置 mapred-site.xml

mapred-site.xml 用于设置监控 Map 与 Reduce 程序的 JobTracker 任务分配情况以及 TaskTracker 任务运行情况。Hadoop 提供了设置的模板文件，可以自行复制修改。在“终端”程序中输入下列命令：

> **复制模板文件：由 mapred-site.xml.template 至 mapred-site.xml**

```
sudo cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/
etc/hadoop/mapred-site.xml
```

执行过程如图 4-25 所示。

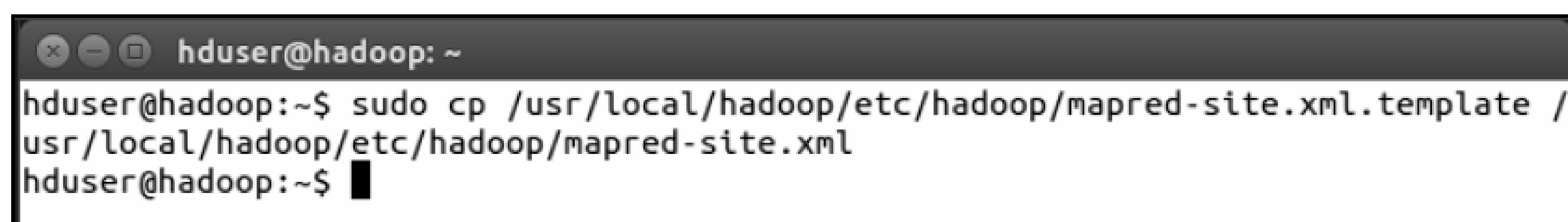


图 4-25　设置 mapred-site.xml

> **编辑 mapred-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

输入后按 Enter 键就会打开 mapred-site.xml，输入图 4-26 所示的内容。

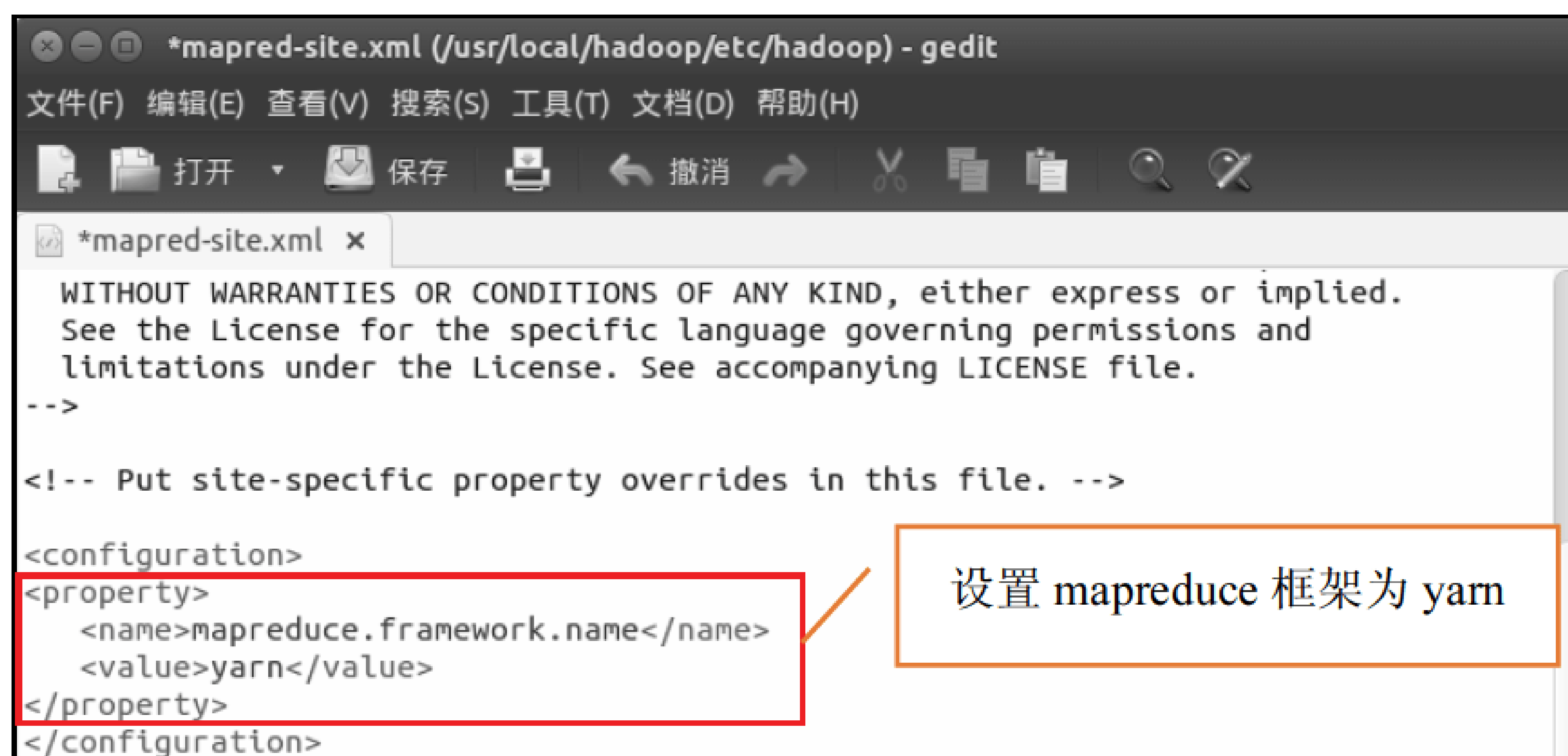


图 4-26　设置 mapred-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 05　设置 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统，在“终端”程序中输入下列命令：

➢ **编辑 hdfs-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，请在 <configuration> </configuration> 标签之间输入如图 4-27 所示的内容。

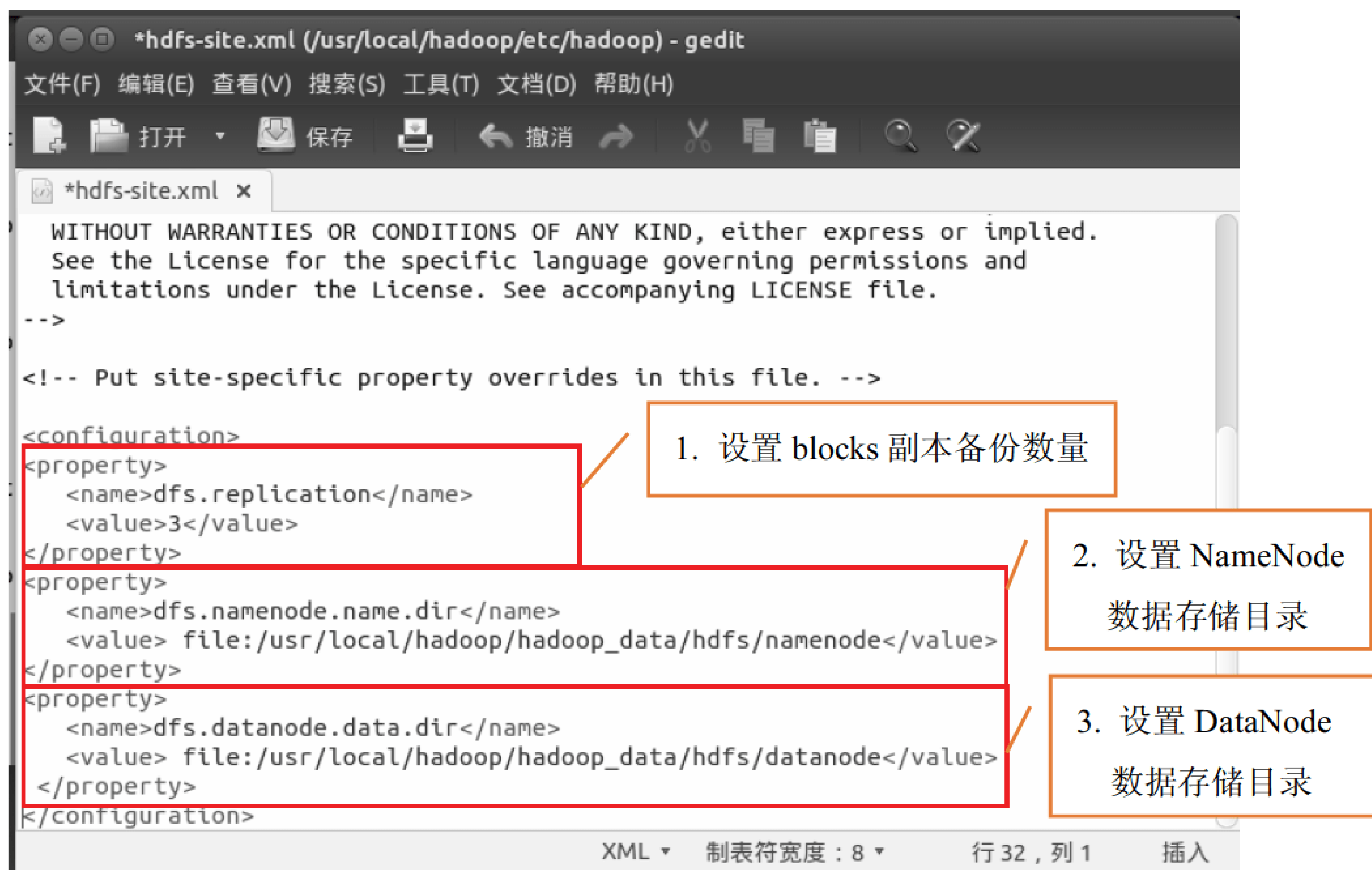


图 4-27　设置 hdfs-site.xml

默认的 blocks 副本备份数量是每一个文件在其他 node 的备份数量，默认值为 3（可参考第 1.3 节的说明）。编辑完成后，先保存，再关闭 gedit。

4.6 创建并格式化 HDFS 目录

步骤 01 创建 namenode、datanode 数据存储目录

在“终端”程序中输入下列命令，创建 namenode、datanode 数据存储目录：

➢ **创建 namenode 数据存储目录**

```
sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

➢ **创建 datanode 数据存储目录**

```
sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➢ **将 Hadoop 目录的所有者更改为 hduser**

```
sudo chown hduser:hduser -R /usr/local/hadoop
```

Linux 是多人多任务的操作系统，所有的目录或文件都具有所有者，使用 chown 可以将目录或文件的所有者更改为 hduser。

执行结果界面如图 4-28 所示。

```
hduser@hadoop: ~
hduser@hadoop:~$ sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
hduser@hadoop:~$ sudo mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
hduser@hadoop:~$ sudo chown hduser:hduser -R /usr/local/hadoop
hduser@hadoop:~$
```

图 4-28　创建 namenode、datanode 数据存储目录

步骤 02 格式化 namenode

在“终端”程序中输入下列命令：

➢ **将 HDFS 进行格式化**

```
hadoop namenode -format
```

执行结果界面如图 4-29 所示。

提示

如果 HDFS 已经有数据，可以执行 HDFS 格式化命令：

```
hadoop namenode -format
```

这个操作会删除所有的数据。

```
hduser@hadoop: ~
hduser@hadoop:~$ hadoop namenode -format
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

15/04/28 20:58:22 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = hadoop/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.6.0
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/
hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/jac
kson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/
usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/s
hare/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/usr/local/hadoop/share/hadoo
p/common/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore
-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:
/usr/local/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/usr/local/had
oop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/ha
doop/common/lib/curator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/commo
n/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.j
ar:/usr/local/hadoop/share/hadoop/common/lib/jersey-server-1.9.jar:/usr/local/ha
doop/share/hadoop/common/lib/apacheds-i18n-2.0.0-M15.jar:/usr/local/hadoop/share
/hadoop/common/lib/commons-el-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/
```

图 4-29　格式化 namenode

4.7 启动 Hadoop

在之前的步骤中已经完成了 Hadoop Single Node Cluster 的安装，现在开始启动 Hadoop。可以使用以下两种方式：

- 分别启动 HDFS、YARN，使用 start-dfs.sh（启动 HDFS）、start-YARN.sh（启动 YARN）。
- 同时启动 HDFS、YARN，使用 start-all.sh。

步骤 01　启动 HDFS

在“终端”程序中输入下列命令：

➢ **启动 HDFS**

```
start-dfs.sh
```

执行结果的屏幕显示界面如图 4-30 所示，第 1 次执行会询问是否继续联机，请按下yes。

```
hduser@hadoop: ~
hduser@hadoop:~$ start-dfs.sh
Starting namenodes on [localhost]
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is d1:ae:5f:58:8f:26:34:7d:51:ec:05:61:b3:0b:42:22.
Are you sure you want to continue connecting (yes/no)? yes
localhost: Warning: Permanently added 'localhost' (ECDSA) to the list of known h
osts.
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-na
menode-hadoop.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-da
tanode-hadoop.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is d1:ae:5f:58:8f:26:34:7d:51:ec:05:61:b3:0b:42:22.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts
.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-hadoop.out
hduser@hadoop:~$
```

启动 namenode

启动 datanode

图 4-30　启动 HDFS

步骤02 启动 Yarn

在“终端”程序中输入下列命令：

➢ **启动 Hadoop MapReduce 框架 Yarn**

```
start-yarn.sh
```

执行结果界面如图 4-31 所示：第 1 次执行会询问是否继续连接，请按下 yes。

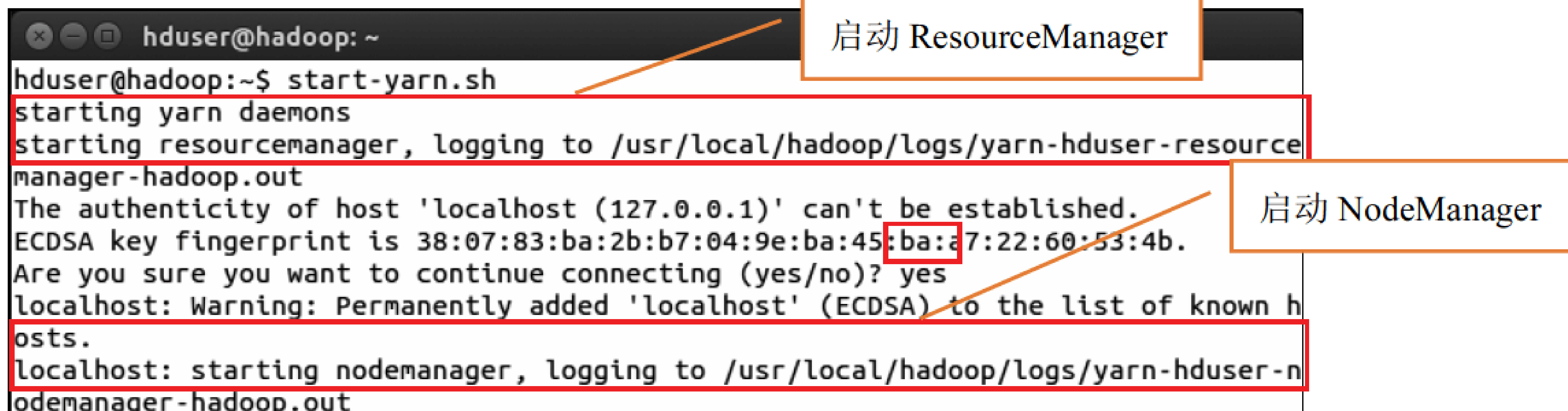


图 4-31　启动 Yarn

步骤03 同时启动 HDFS 和 Yarn：start-all.sh

另外，还可以使用 start-all.sh。

➢ **同时启动 HDFS、Yarn**

```
start-all.sh
```

执行结果的屏幕显示界面如图 4-32 所示。

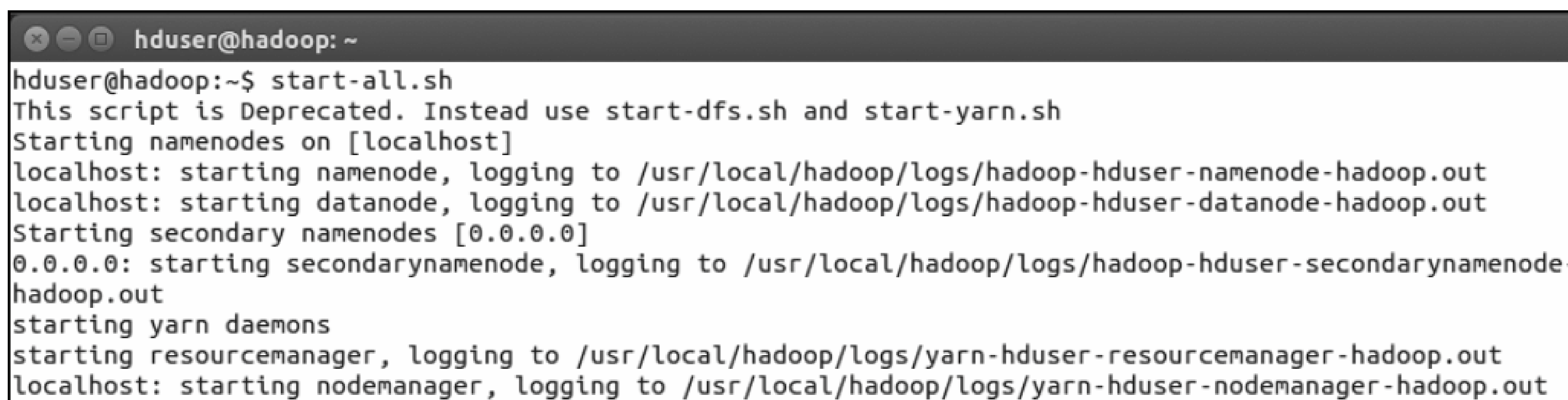


图 4-32　同时启动 HDFS 和 Yarn

步骤04 使用 jps 查看已经启动的进程

jps（Java Virtual Machine Process Status Tool），可以查看当前所运行的进程（process），在“终端”程序中输入下列命令：

➢ **查看 NameNode、DataNode 进程是否启动**

```
jps
```

运行后屏幕显示界面如图 4-33 所示。

```
hduser@hadoop:~$ jps
11897 ResourceManager
12018 NodeManager
12311 Jps
11761 SecondaryNameNode
11447 NameNode
11589 DataNode
```

Yarn 相关进程

HDFS 相关进程

图 4-33 查看已经启动的进程

以上启动进程的说明如下：因为只有一台服务器，所以所有功能都集中在这台服务器中，我们可以看到以下内容。

- **HDFS 功能：** NameNode、SecondaryNameNode、DataNode 已经启动。
- **MapReduce2（YARN）：** ResourceManager、NodeManager 已经启动。

4.8 打开 Hadoop Resource-Manager Web 界面

Hadoop ResourceManager Web 界面可用于查看当前 Hadoop 的状态：Node 节点、应用程序、进程运行状态。

步骤 01 打开 Hadoop ResourceManager Web 界面

打开浏览器 Firefox，在网址栏输入：

➢ Hadoop ResourceManager Web **界面网址**

```
http://localhost:8088/
```

参照下列步骤（见图 4-34），就可以看到屏幕显示界面。

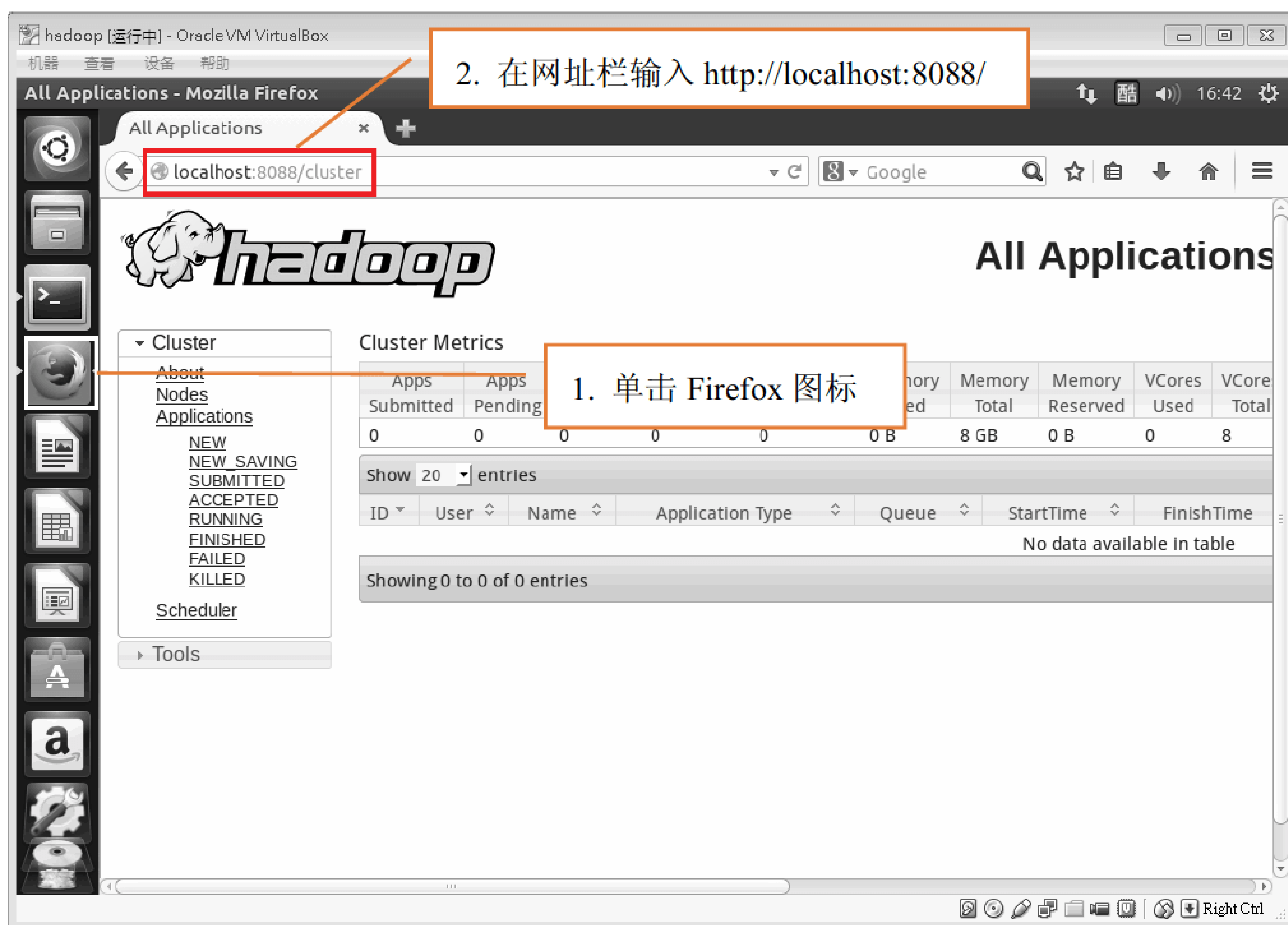


图 4-34　打开 Hadoop ResourceManager Web 界面

步骤 02 查看已经运行的节点 Nodes

单击 Nodes 时，会显示当前的节点。不过，因为我们安装的是 Single Node Cluster，所以当前只有一个节点，如图 4-35 所示。



图 4-35　查看已经运行的节点

4.9 NameNode HDFS Web 界面

HDFS Web 界面可以检查当前 HDFS 与 DataNode 的运行情况。

步骤 01 打开 NameNode HDFS Web 界面

打开浏览器 Firefox，在网址栏输入：

➢ **HDFS Web UI 网址**

```
http://localhost:50070/
```

可以看到屏幕显示界面，如图 4-36 所示。

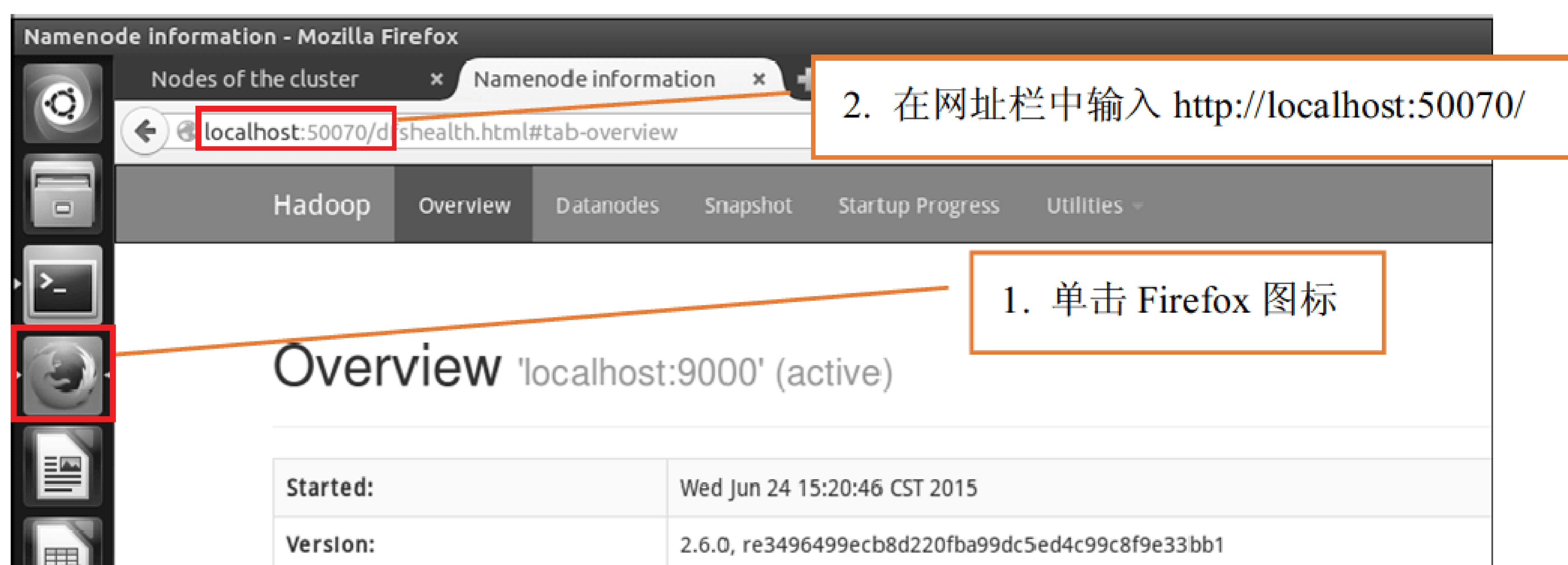


图 4-36 打开 NameNade HDFS Web 界面

步骤 02 查看 Live Nodes

向下浏览，可以看到当前启动了 1 个节点，如图 4-37 所示。

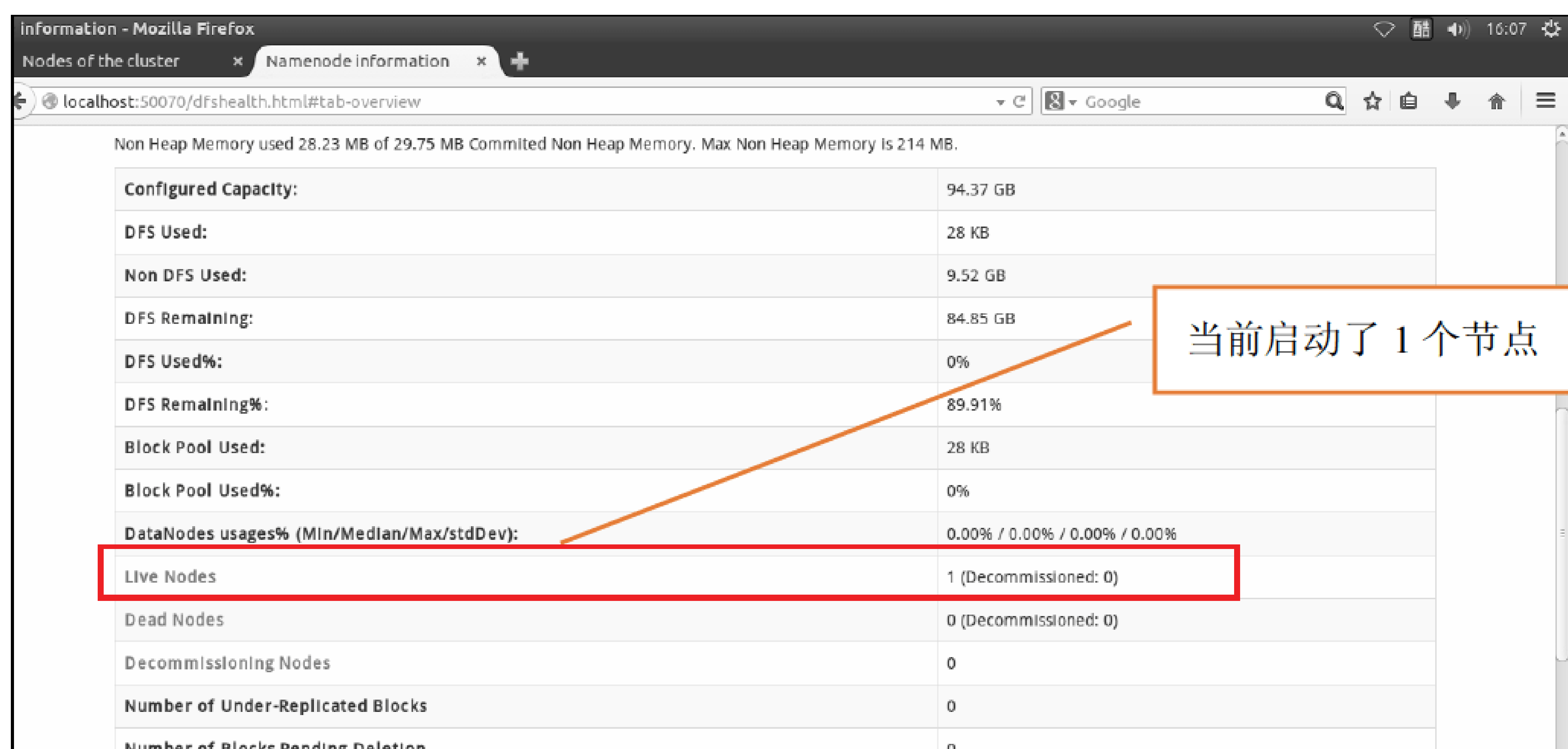


图 4-37 查看 Live Nodes（启动的节点数）

步骤 03 查看 DataNodes（见图 4-38）

图 4-38　查看 DataNodes

4.10 结论

本章我们介绍了如何安装 Hadoop Single Node Cluster。因为只有一台服务器，所有功能都集中在一台服务器中，所以无法发挥分布式计算与存储的功能。下一章我们将介绍如何安装由多台计算机组成的 Hadoop Multi Node Cluster。

第 5 章 Hadoop Multi Node Cluster的安装

本章将介绍如何安装多台机器的 Hadoop 集群（Hadoop Multi Node Cluster），以及 Hadoop 资源管理（ResourceManager）与 NameNode HDFS Web 界面。

Hadoop Multi Node Cluster 规划如图 5-1 所示，由多台计算机组成：

- 有一台主要的计算机 master，在 HDFS 担任 NameNode 角色、在 MapReduce2（YARN）担任 ResourceManager 角色。
- 有多台辅助计算机 data1、data2、data3，在 HDFS 担任 DataNode 角色、在 MapReduce2（YARN）担任 NodeManager 角色。

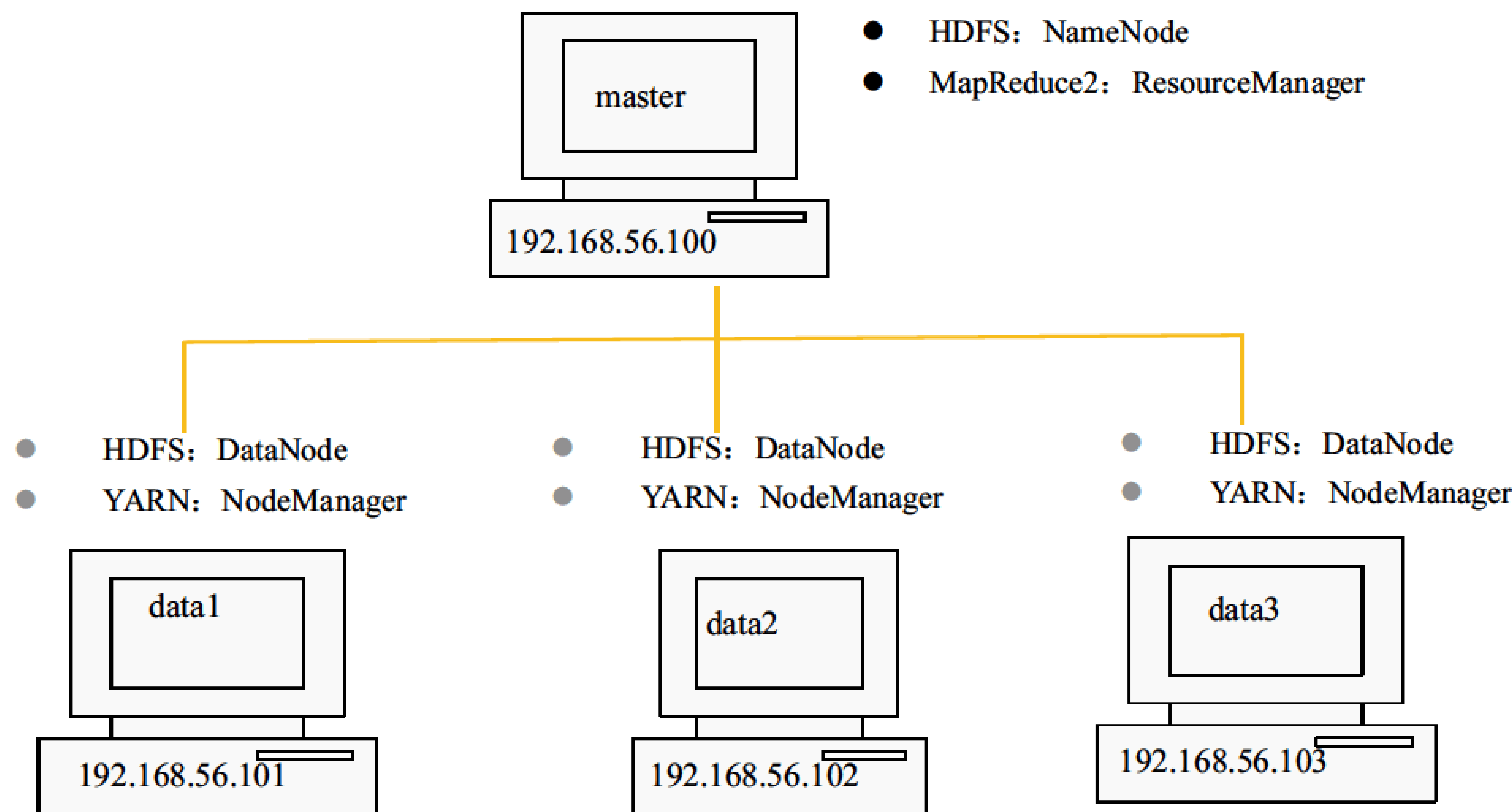


图 5-1　Hadoop Multi Node Cluster 规划图

与图 5-1 对应的 Hadoop Multi Node Cluster 规划说明如表 5-1 所示。

表 5-1　规划表

服务器名称	内部 IP	HDFS	YARN
master	192.168.56.100	NameNode	ResourceManager
data1	192.168.56.101	DataNode	NodeManager
data2	192.168.56.102	DataNode	NodeManager
data3	192.168.56.103	DataNode	NodeManager

图 5-1 的 Hadoop Multi Node Cluster 架构必须有 4 台实体服务器才能建立，以发挥多台计算机并行处理的优势。

考虑到大部分读者只有一台个人计算机，为了方便大家上机演练，这里我们使用 VirtualBox 创建 4 台虚拟主机 master、data1、data2、data3。因为是虚拟主机，所以无法享受到多台计算机并行处理的优势。不过，你在虚拟主机所学到的设置，完全可以用于创建实体主机 Hadoop Multi Node Cluster。

➢ Hadoop Multi Node Cluster 的安装步骤（见表 5-2）

表 5-2　安装步骤说明

顺序	安装步骤	说明
1	复制 Single Node Cluster 到 data1	为了节省安装时间，我们将第 4 章所创建的 Single Node Cluster Hadoop 复制到 data1，创建 data1 虚拟机
2	VirtualBox 适配卡设置	设置两块适配卡 ● 适配卡 1：设置为“NAT 适配卡”，可以通过 Host 主机连接至外部网络 ● 适配卡2：设置为“内部网络适配器”，用于建立内部网络
3	设置 data1 服务器	设置 Multi Node Cluster 服务器，配置设置文件共同的部分（固定 IP、hostname、core-site.xml、YARN-site.xml、mapred-site.xml、hdfs-site.xml）
4	复制 data1 服务器至 data2、data3、master	我们已经设置了 data1 服务器、配置设置文件共同的部分，为了节省安装时间，所以复制 data1 到 data2、data3、master
5	设置 data2 服务器	设置 data2 固定 ip、hostname
6	设置 data3 服务器	设置 data3 固定 ip、hostname
7	设置 master 服务器	设置 NameNode 服务器（master）：设置固定 IP、hostname、hdfs-site.xml、masters、slaves
8	master 连接到 data1、data2、data3 创建 HDFS 目录	重新启动 master 与 data1、data2、data3 后，master 通过 SSH 连接 data1、data2、data3 并创建 HDFS 目录
9	建立与格式化 NameNode HDFS 目录	创建 NameNode HDFS 目录，并且格式化 HDFS 目录
10	启动 Hadoop Multi Node cluster	启动 Hadoop Multi Cluster 并使用 jps 查看当前所运行的进程
11	打开 Hadoop Resource Manager Web 界面	Hadoop ResourceManagerWeb 界面可用于查看当前 Hadoop:Node 节点、应用程序、进程运行状态
12	打开 NameNode Web 界面	HDFS Web 界面，可以检查当前 HDFS 与 DataNode 运行情况

➢ Hadoop 2.6 Multi Node Cluster 安装命令

对于安装过程中所需要输入的命令，我们已整理在本书有关的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可以节省打字的时间，也不用担心打错字（如果无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序进行复制/粘贴操作，请参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

5.1 把 Single Node Cluster 复制到 data1

为了节省安装时间，我们将复制第 4 章所创建的 Single Node Cluster 来创建 data1 虚拟机。如果不使用复制虚拟机的方法，也可以重复第 4 章的步骤来创建 data1。

步骤 01 复制 Hadoop 到 data1（见图 5-2）

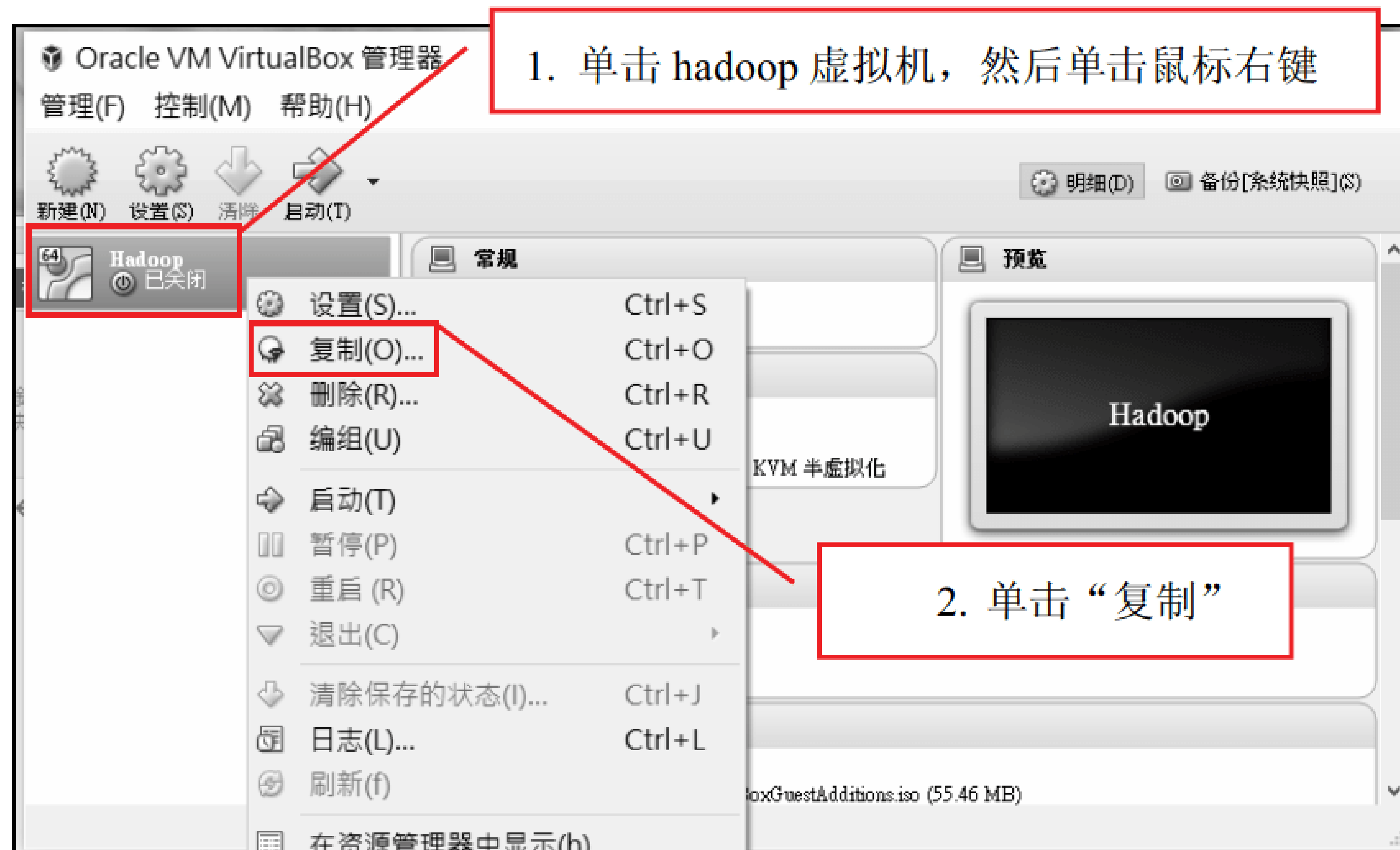


图 5-2　复制 Hadoop 到 data1

步骤 02 设置虚拟机名称（见图 5-3）

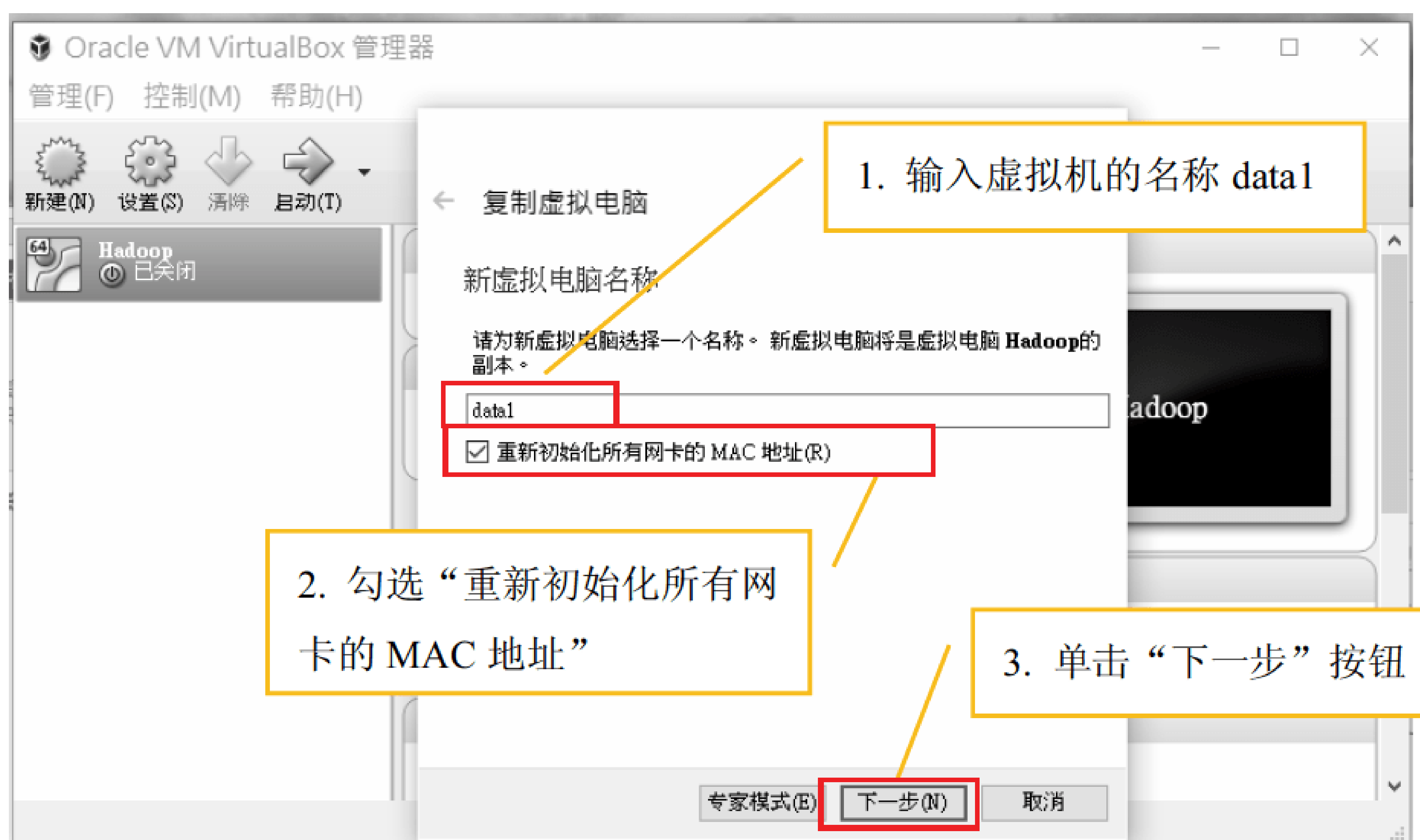


图 5-3　设置虚拟机名称

步骤03 设置复制类型

先参照图 5-4 所示的步骤选择复制类型，再单击“复制”按钮，需要等待几分钟。

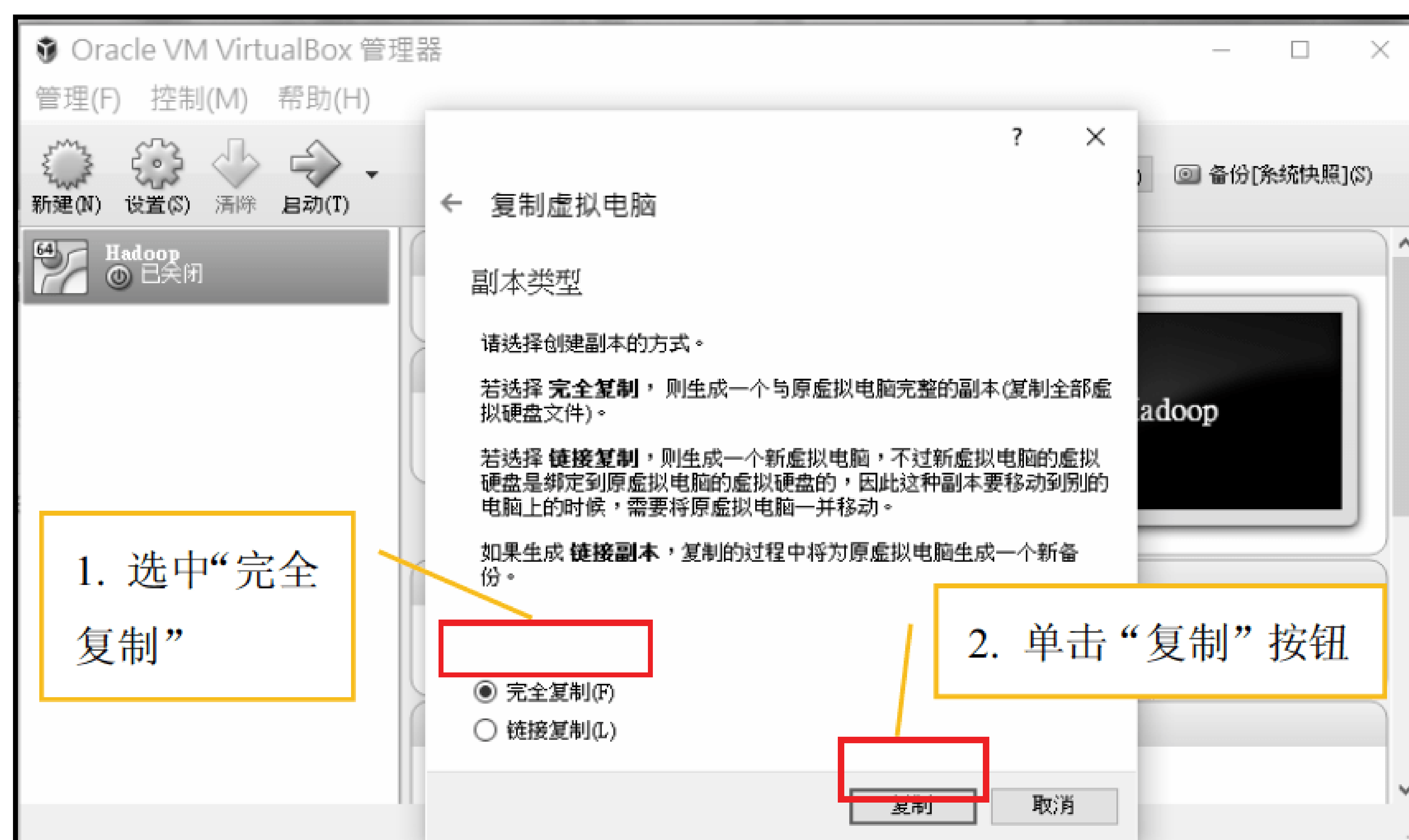


图 5-4 设置复制类型

步骤04 复制完成

复制完成后就会出现 data1，如图 5-5 所示。



图 5-5 复制完成

5.2 设置 VirtualBox 网卡

设置 VirtualBox 网卡，如图 5-6 所示。

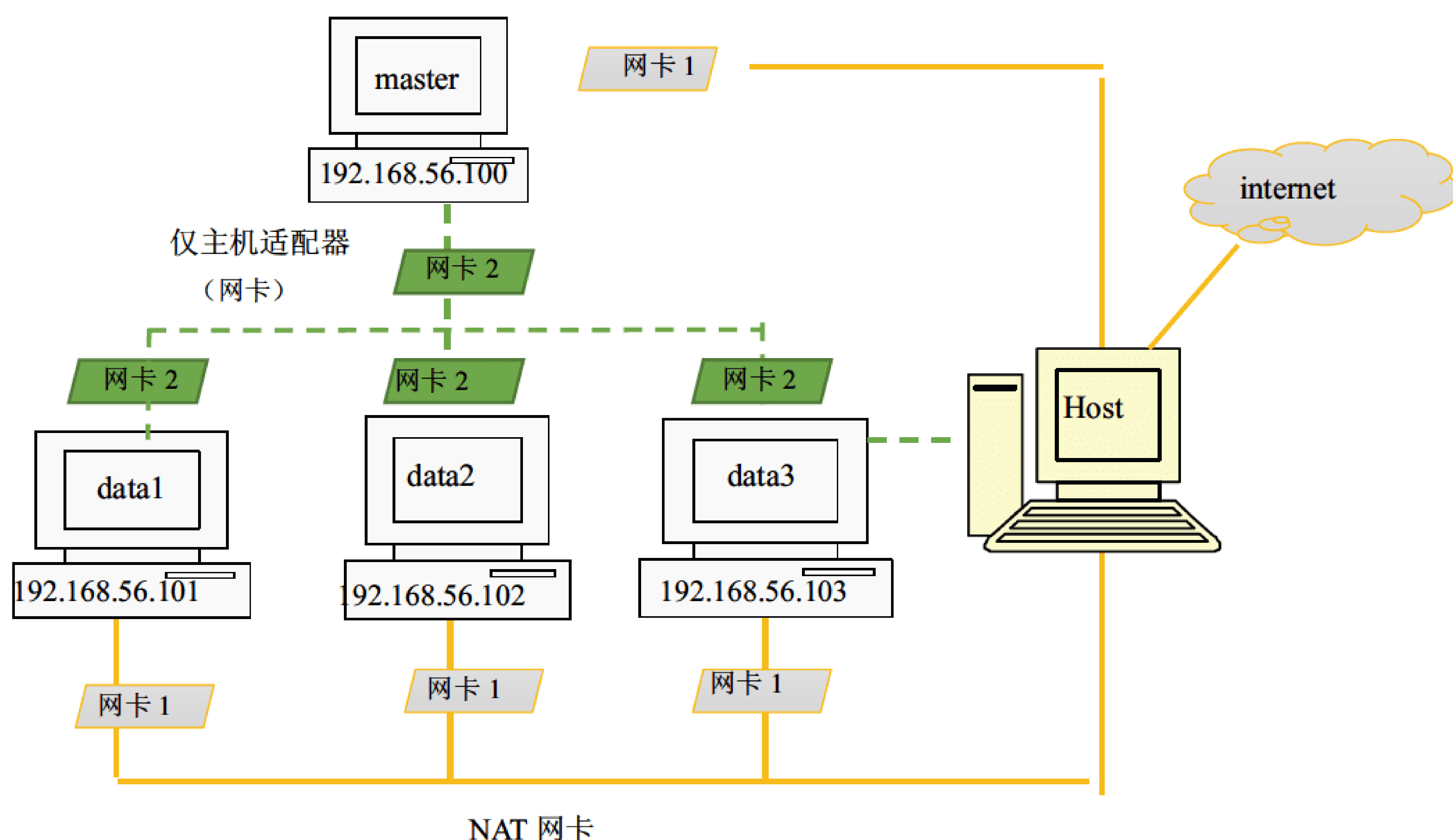


图 5-6　设置网卡示意图

（1）最右边是 Host 主机，就是我们安装 VirtualBox 的主机，本书范例安装的是 Windows 系统，在 VirtualBox 上我们将创建 4 台虚拟主机，分别是 master、data1、data2、data3。

（2）我们将在每一台虚拟主机上设置 2 张网卡。

- 网卡 1：设置为“NAT 网卡”，可以通过 Host 主机连接到外部网络（internet）。
- 网卡 2：设置为“仅主机适配器”（这里的适配器指的是网卡），用于创建内部网络，内部网络连接虚拟主机（master、data1、data2、data3）与 Host 主机。

步骤 01 设置网卡

请参照图 5-7 所示的步骤设置 data1 网卡。

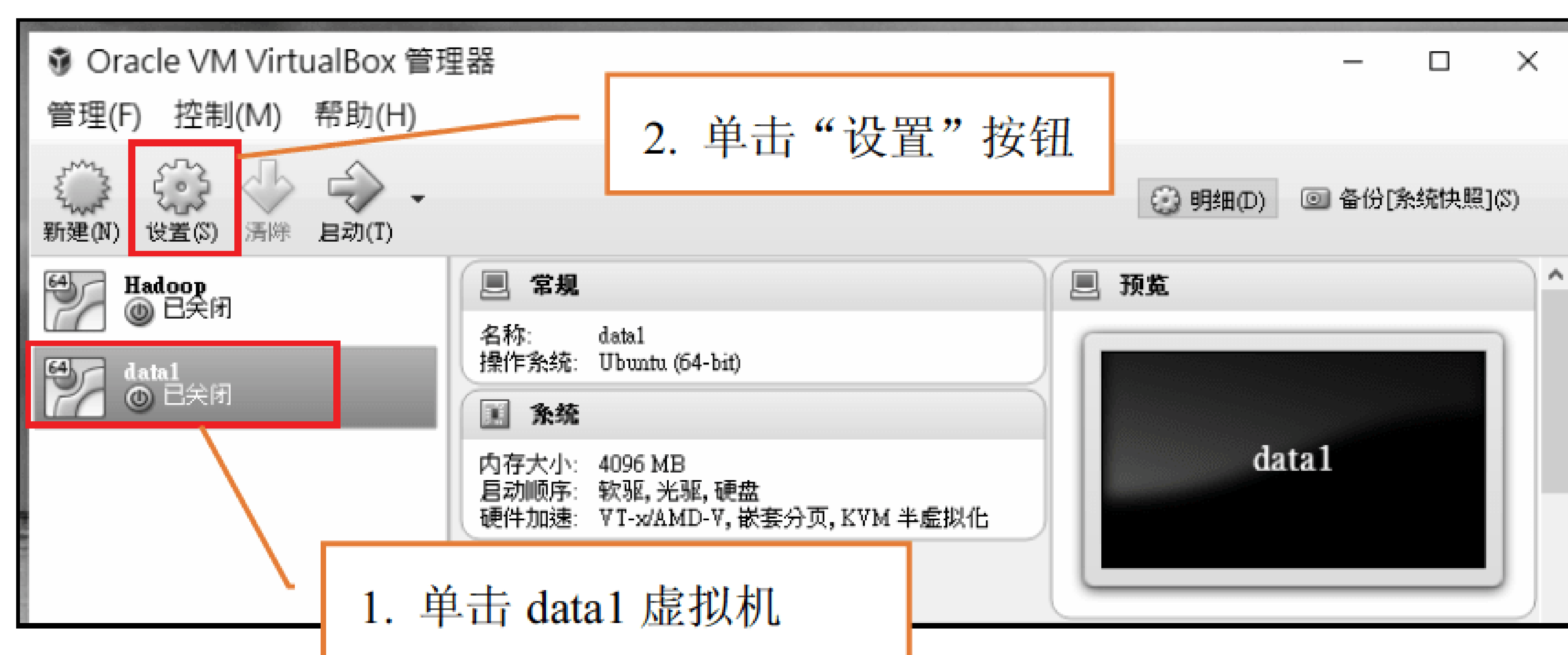


图 5-7　设置 data1 网卡

步骤02 设置网卡 1：“NAT”网卡

设置网卡 1 为“NAT”网卡，这样虚拟主机就可以通过 Host 主机连接至外部网络（internet），具体设置步骤如图 5-8 所示。

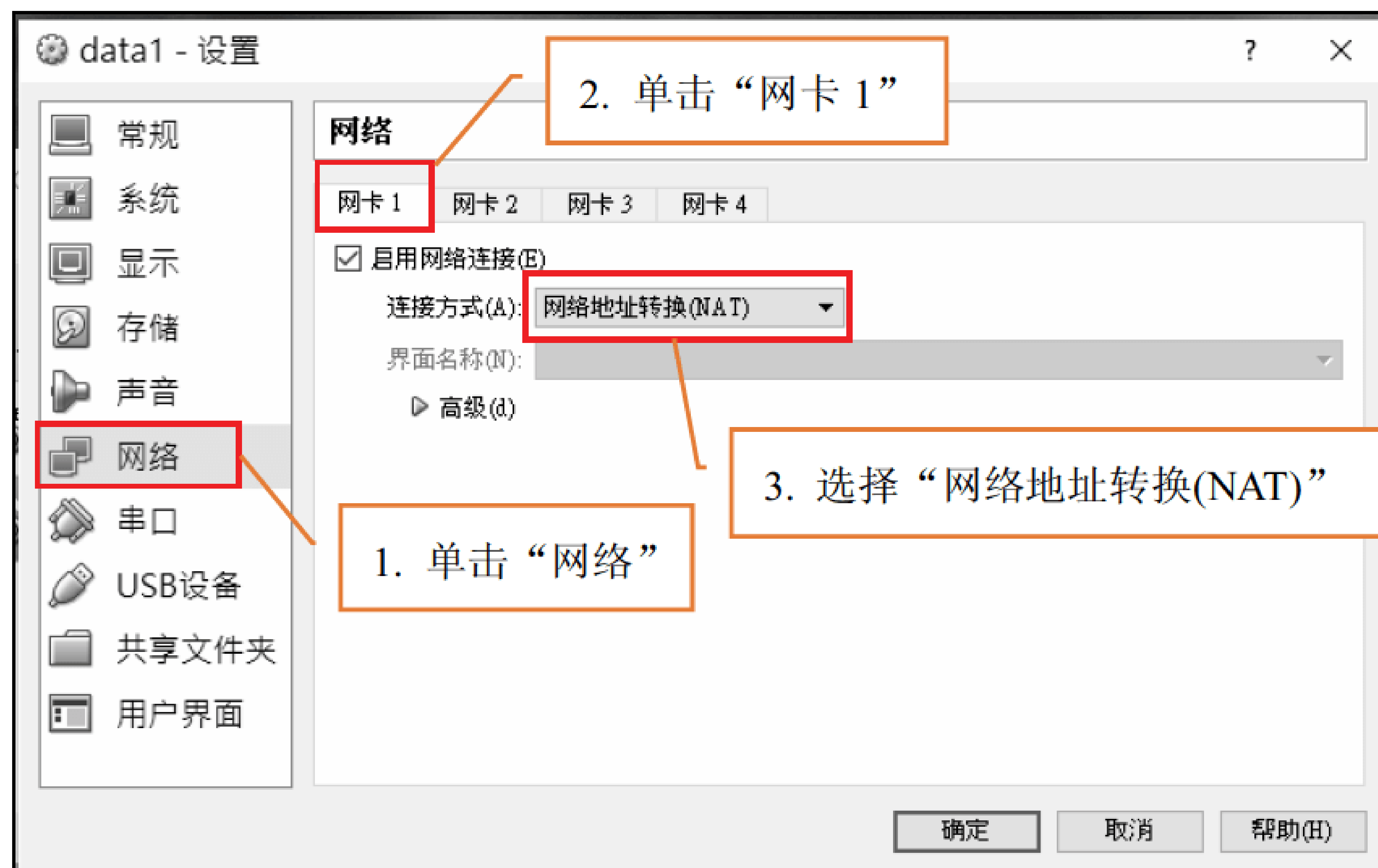


图 5-8　设置网卡 1

步骤03 设置网卡 2：仅主机适配器（网卡）

设置网卡 2 为“仅主机适配器”（网卡），用于建立内部网络，内部网络可连接虚拟主机（master、data1、data2、data3）与 Host 主机。具体设置步骤如图 5-9 所示。

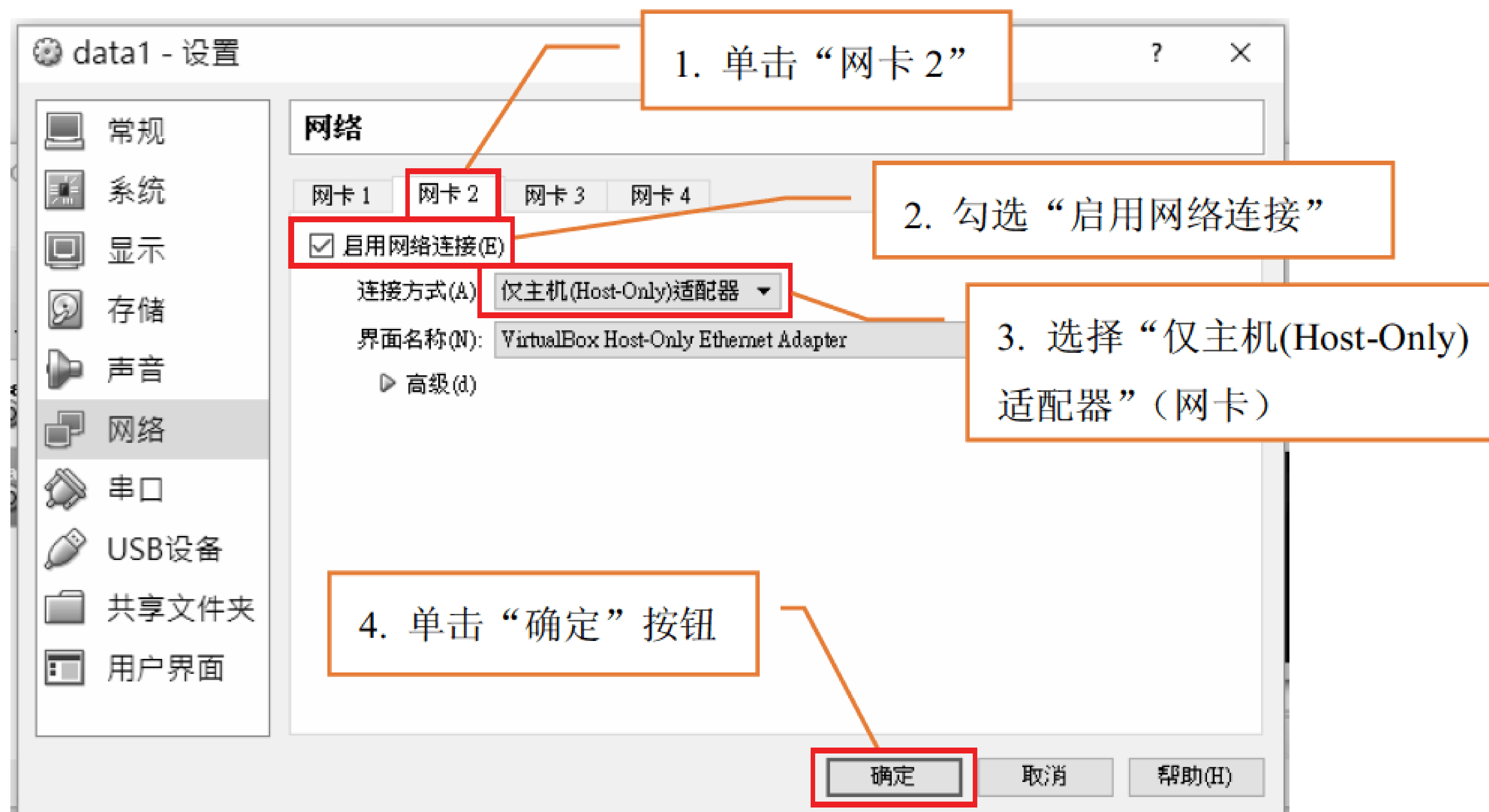


图 5-9　设置网卡 2

目前只设置 data1，后续我们会将 data1 复制至 data2、data3、master，也会将这些设置复制过去。

5.3 设置 data1 服务器

设置 Multi Node Cluster 服务器，配置设置文件共同的部分：固定 IP、hostname、core-site.xml、YARN-site.xml、mapred-site.xml、hdfs-site.xml。

步骤 01 启动 data1 虚拟机

复制完成后就会出现 data1。单击 data1，然后单击“启动”按钮来启动虚拟机，如图 5-10 所示。



图 5-10　启动 data1 虚拟机

步骤 02 编辑网络配置文件设置固定 IP

我们必须设置 data1 虚拟主机每次开机都是使用固定 IP：192.168.56.101，请在 data1 的“终端”程序中输入下列命令：

➢ **编辑 interfaces 网络配置文件**

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-11 所示。

编辑完成后，先保存，再关闭 gedit。

以上设置说明如下：

➢ **设置网卡 1**

设置为“NAT 网卡”，可以通过 Host 主机连接至外部网络（internet），设置为 eth0，并设置 dhcp 自动获得 IP 地址。

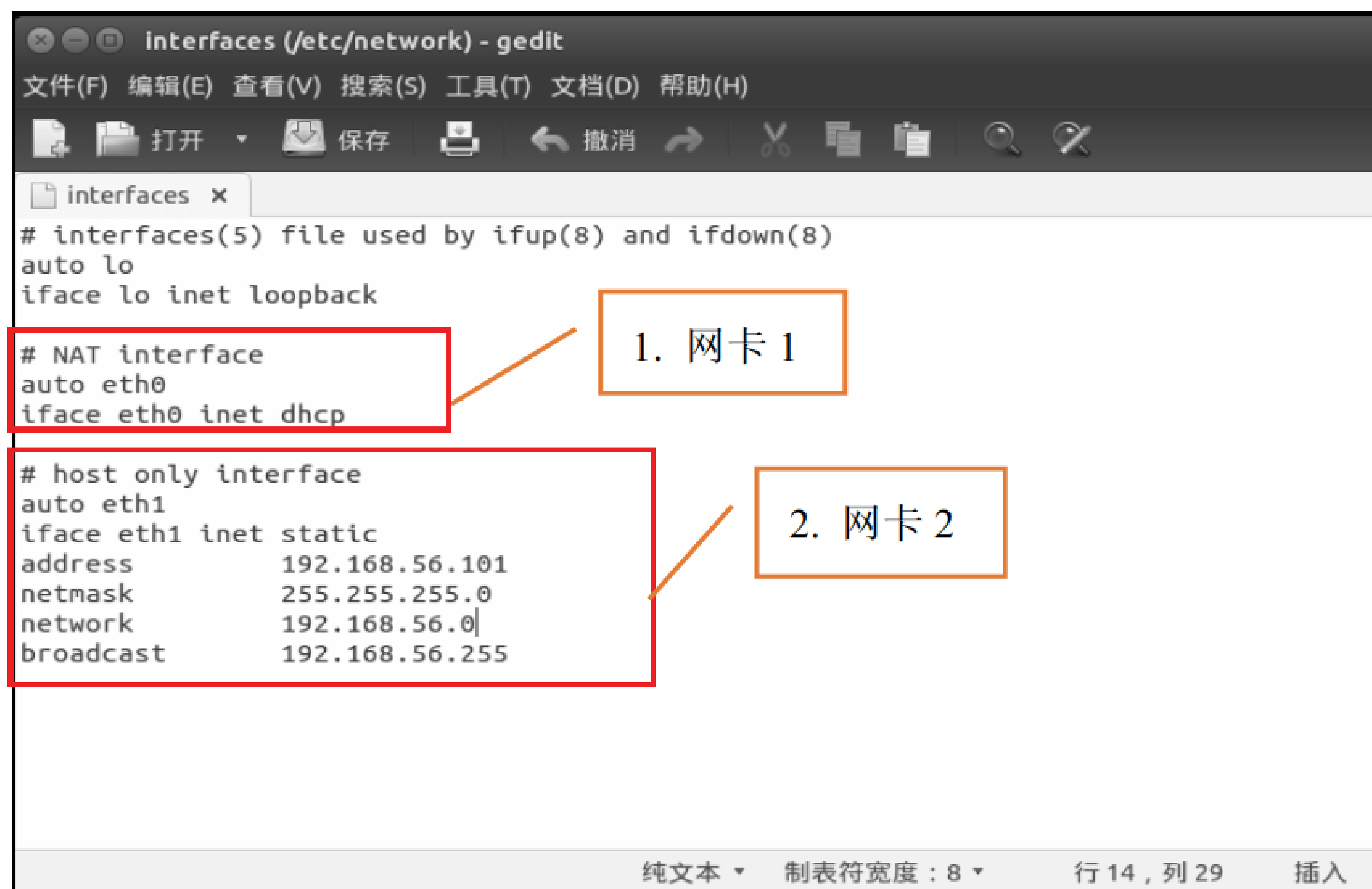


图 5-11 编辑 interfaces 网络配置文件

➢ 设置网卡 2

设置为“仅主机适配器”，用于建立内部网络，内部网络连接虚拟主机（master、data1、data2、data3）与 Host 主机。设置为 eth1，并设置为 static，即指定固定 IP 地址。

步骤 03 设置 hostname

然后必须设置 data1 的 hostname（主机名）。
在 data1 的“终端”程序中输入下列命令：

➢ 编辑 hostname 主机名

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 显示界面，如图 5-12 所示。



图 5-12 编辑 hostname 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04 设置 hosts 文件

我们建立的 Hadoop Multi Node Cluster 中有 4 台计算机，如何让网络中所有的计算机都知道其他计算机的主机名与 IP 呢？可以编辑 hosts 文件或设置 DNS。hosts 文件通常用于补充或取代网络中 DNS 的功能，和 DNS 不同的是，计算机的用户可以直接对 hosts 文件进行控制。hosts 文件可存储计算机网络中各节点的信息，负责将主机名映射到对应的 IP 地址（名字解析）。

请在 data1“终端”程序中输入下列命令：

➢ **编辑 hosts 文件**

```
sudo gedit /etc/hosts
```

输入后按 Enter 键就会打开 hosts 文件，请设置各节点的主机名与相对应的 IP 地址，如图 5-13 所示。

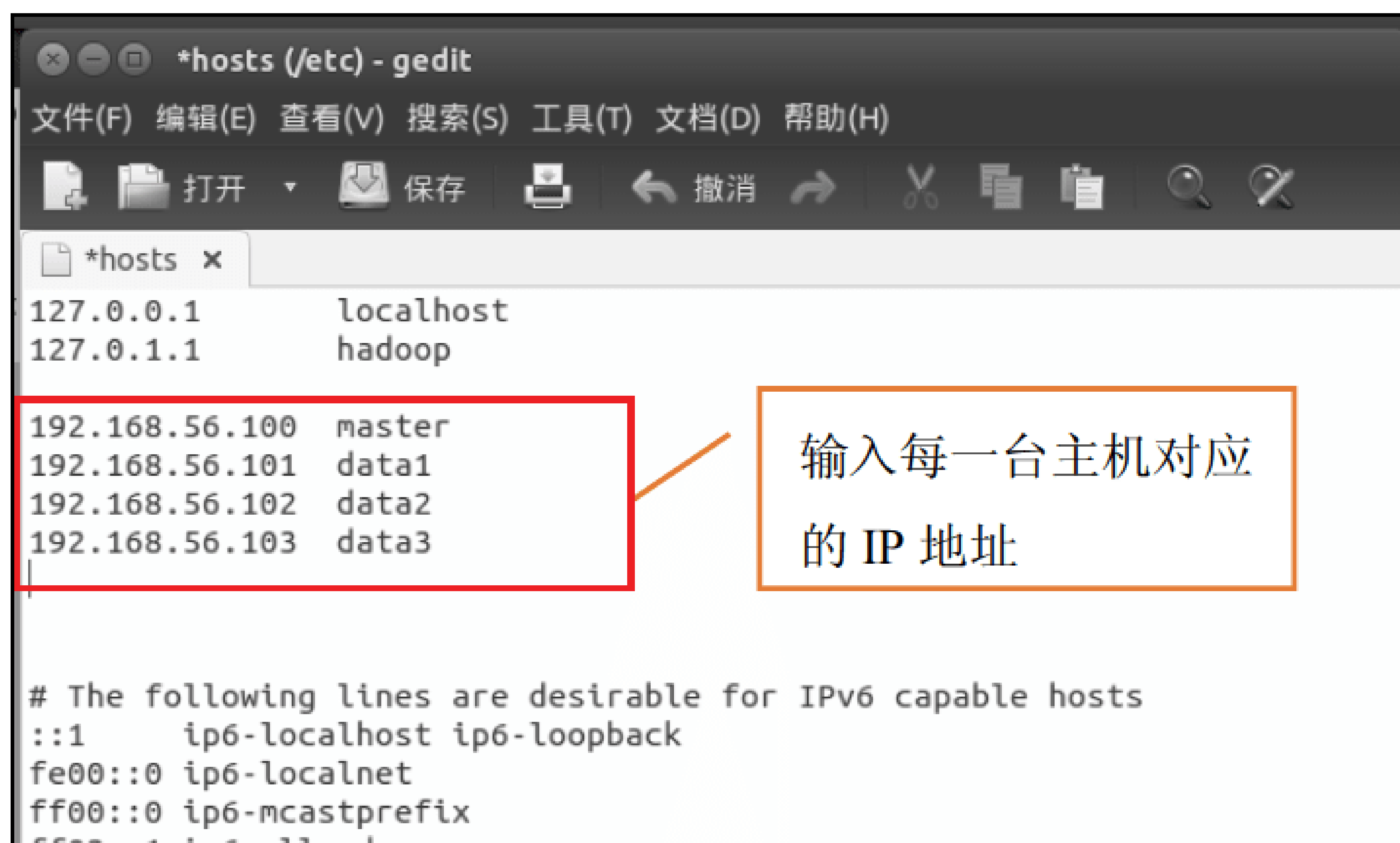


图 5-13 编辑 hosts 文件

编辑完成后，先保存，再关闭 gedit。

步骤 05 编辑 core-site.xml

在 core-site.xml 中，我们必须设置 HDFS 的默认名称，当使用命令或程序来存取 HDFS 时，可使用此名称。之前 Single Node Cluster 因为只有一台计算机，所以我们设置 namenode 位置为 localhost 即可，但是现在有多台计算机，所以必须指定主机名。

请在 data1 的“终端”程序中输入下列命令：

➢ **编辑 core-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml
```

输入后按 Enter 键就会打开 core-site.xml，如图 5-14 所示。

编辑完成后，先保存，再关闭 gedit。之后当使用程序存取 HDFS 时，会使用 hdfs://master:9000 这个目标来存取 HDFS。

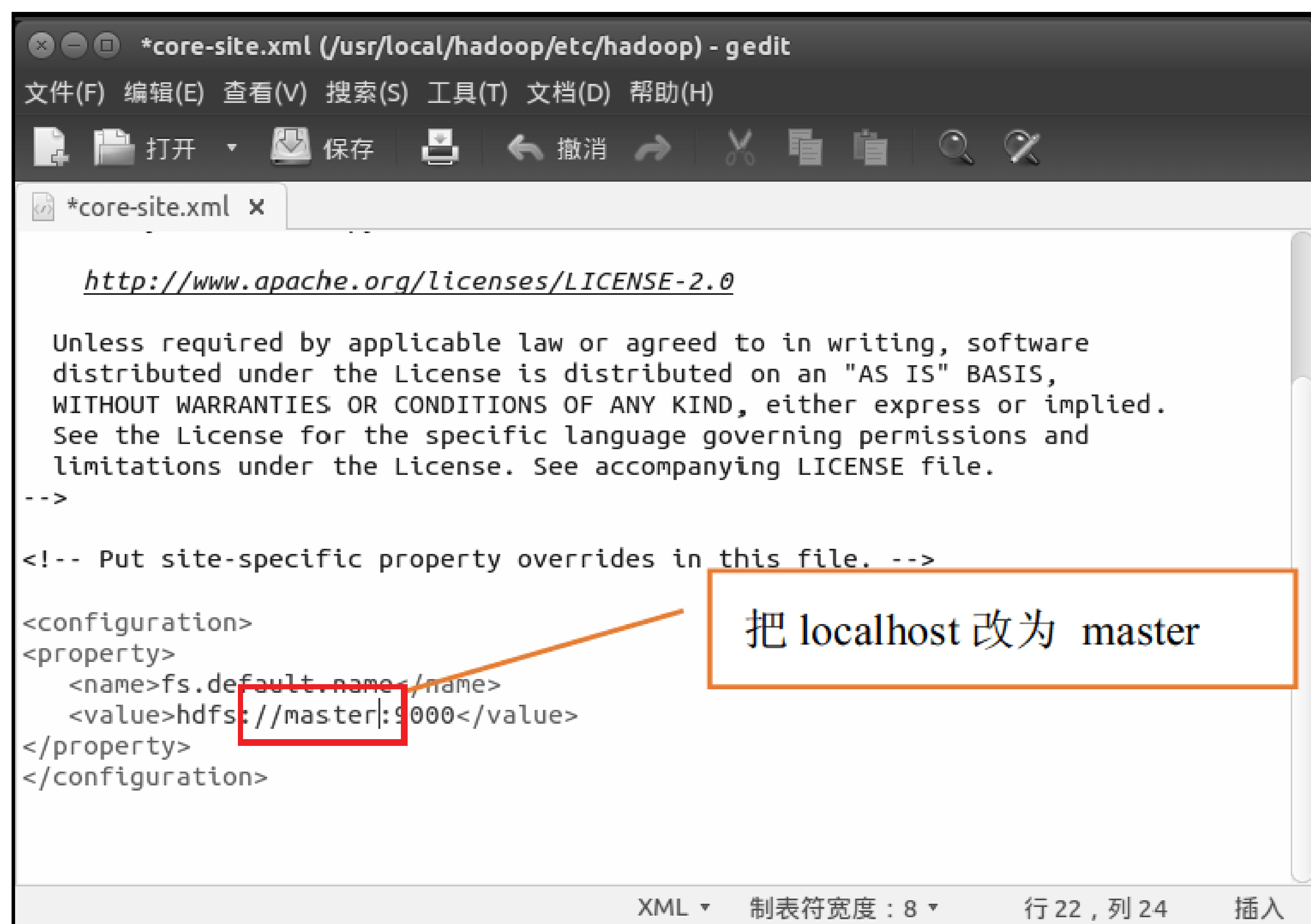


图 5-14 编辑 core-site.xml

步骤06 编辑 YARN-site.xml

YARN-site.xml 文件是 MapReduce2（YARN）相关配置的设置。

在 data1 的“终端”程序中输入下列命令：

➢ **编辑 YARN-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

输入后按 Enter 键就会打开 YARN-site.xml，屏幕显示界面如图 5-15 所示，在其中输入图示中的指令。

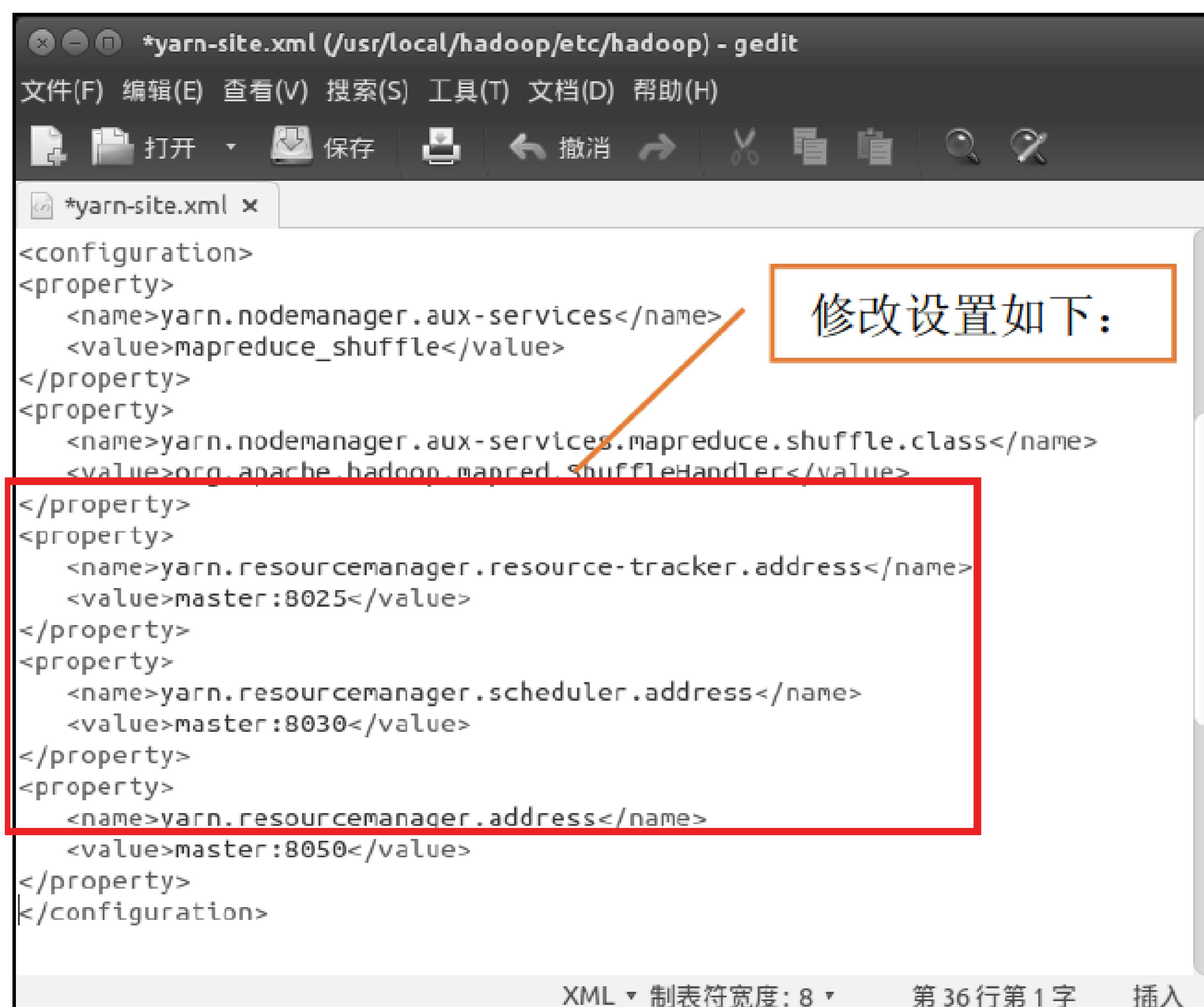


图 5-15 编辑 yarn-site.xml

编辑完成后，先保存，再关闭 gedit。

上述命令设置与说明如下：

- ResourceManager 主机与 NodeManager 的连接地址为 8025。
- ResourceManager 与 ApplicationMaster 的连接地址为 8030。
- ResourceManager 与客户端的连接地址为 8050。

步骤 07 编辑 mapred-site.xml

mapred-site.xml 用于设置监控 Map 与 Reduce 程序的 JobTracker 任务分配情况，以及 TaskTracker 任务运行状况。在 data1 的“终端”程序中输入下列命令：

➢ **编辑 mapred-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

输入后按 Enter 键就会打开 mapred-site.xml，如图 5-16 所示。

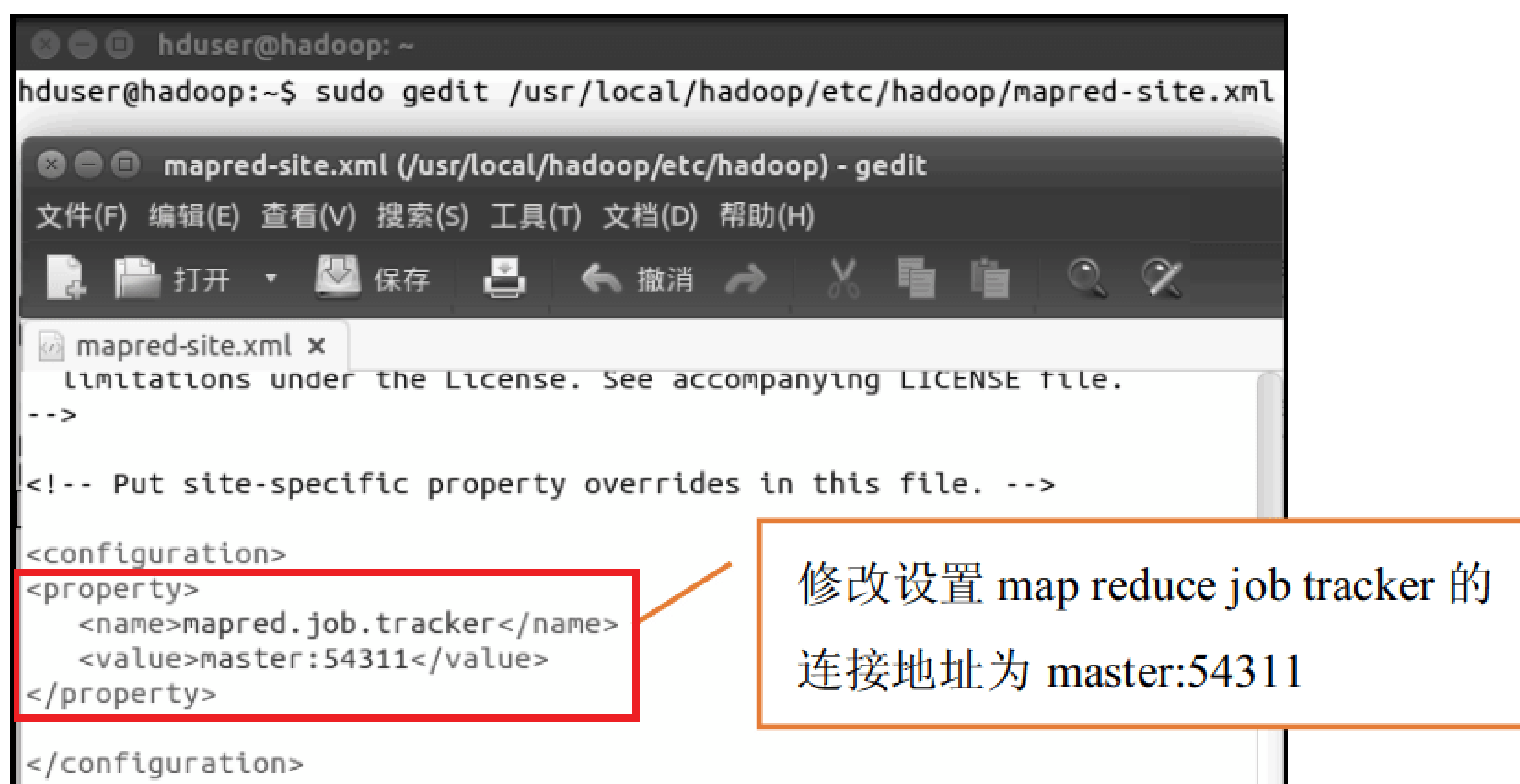


图 5-16　编辑 mapred-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 08 编辑 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统的相关配置。之前在 Single Node Cluster 中因为只有一台服务器，所以同时身兼 NameNode 与 DataNode 角色。但是 data1 现在只是单纯的 DataNode，所以请删除 NameNode 的设置，只保留 DataNode 的设置。

请在 data1 的“终端”程序中输入下列命令：

➢ **编辑 hdfs-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，如图 5-17 所示。



图 5-17 编辑 hdfs-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 09 data1 重新启动（见图 5-18）



图 5-18 重新启动

步骤 10 确认网络设置（见图 5-19）

重新启动后，在 data1 的“终端”程序中输入下列命令，确认网络设置：

```
ifconfig
```

从图 5-19 所示的界面可以看到，有 eth0 与 eth1 两块网卡，并且内部 IP 是 192.168.56.101。

```
hduser@data1: ~
hduser@data1:~$ ifconfig
eth0      Link encap:以太网  硬件地址 08:00:27:45:19:29
          inet 地址:10.0.2.15  广播:10.0.2.255  掩码:255.255.255.0
          inet6 地址: fe80::a00:27ff:fe45:1929/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:29 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:64 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:3769 (3.7 KB)  发送字节:9031 (9.0 KB)

eth1      Link encap:以太网  硬件地址 08:00:27:02:6d:c2
          inet 地址:192.168.56.101  广播:192.168.56.255  掩码:255.255.255.0
          inet6 地址: fe80::a00:27ff:fe02:6dc2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:36 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:0 (0.0 B)  发送字节:6254 (6.2 KB)

lo        Link encap:本地环回
          inet 地址:127.0.0.1  掩码:255.0.0.0
          inet6 地址: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  跃点数:1
          接收数据包:128 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:128 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:0
          接收字节:7904 (7.9 KB)  发送字节:7904 (7.9 KB)

hduser@data1:~$
```

网卡 1

网卡 2

内部 IP 是 192.168.56.101

图 5-19　确认网络设置

步骤 11　确认对外网络连接正常

可以打开浏览器，确认对外网络连接正常，如图 5-20 所示。

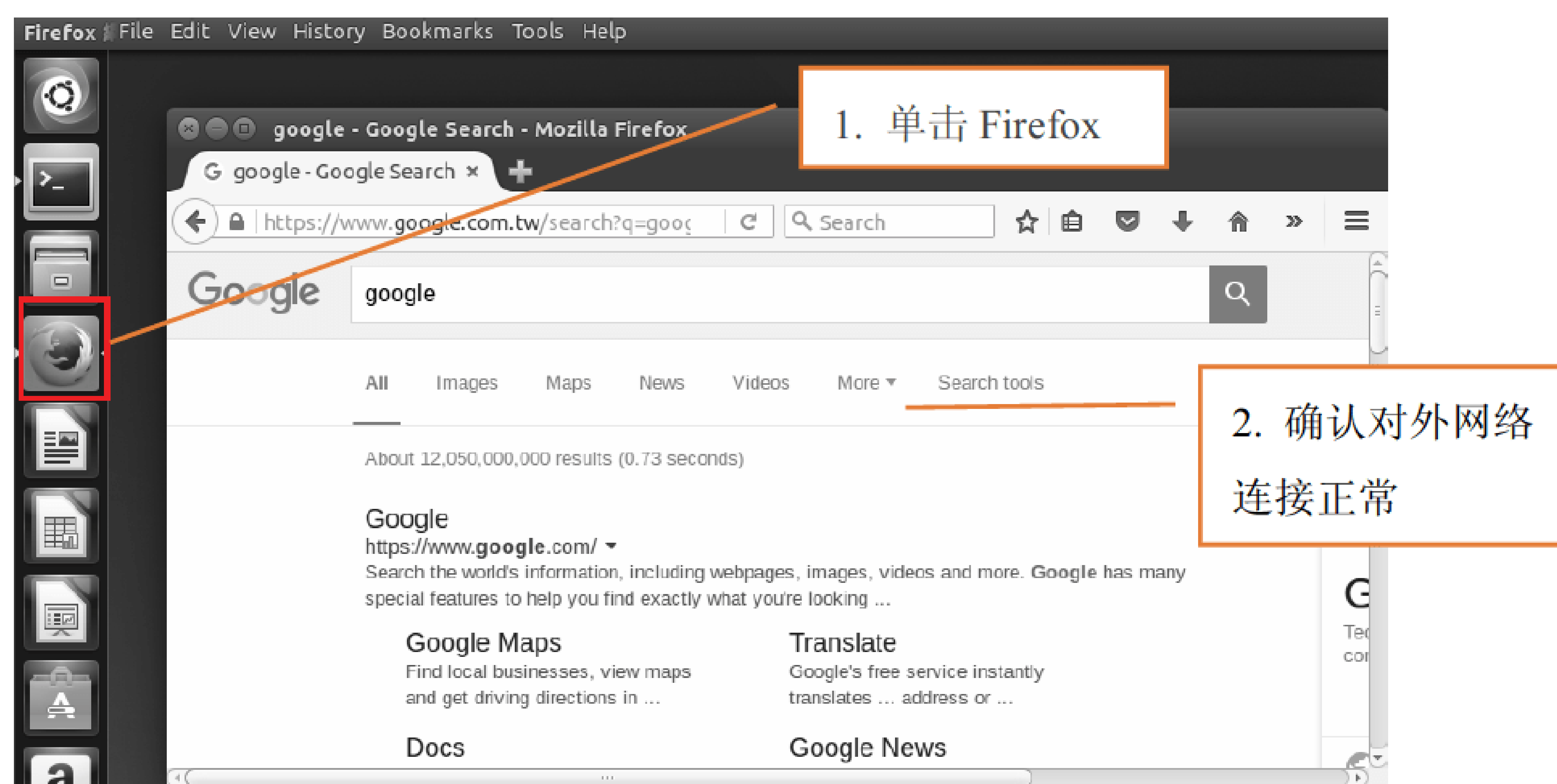


图 5-20　确认对外网络连接正常

步骤 12　data1 虚拟机关机

因为我们后续要将 data1 复制到 data2、data3、master，所以必须先关机，如图 5-21 所示。

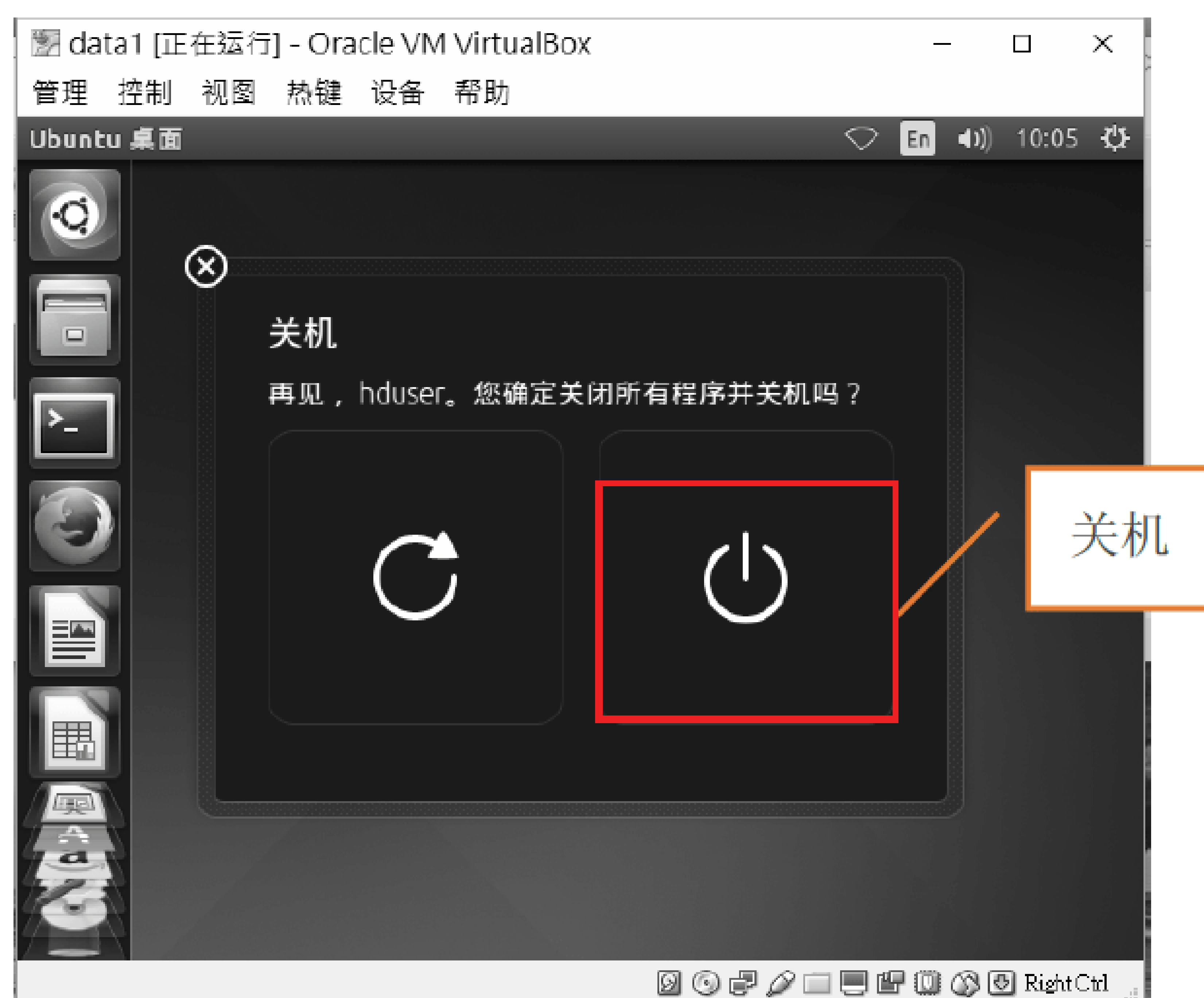


图 5-21 data1 虚拟机关机

5.4 复制 data1 服务器到 data2、data3、master

因为目前 data1 已经设置了 Hadoop Multi Node Cluster 共同的部分，为了节省安装时间，所以复制 data1 到 data2、data3、master。如果不使用虚拟机，也可以重复之前的步骤来创建 data2、data3、master。

步骤 01 将 data1 复制到 data2（见图 5-22）



图 5-22 复制 data1

步骤 02 输入虚拟机名称 data2（见图 5-23）

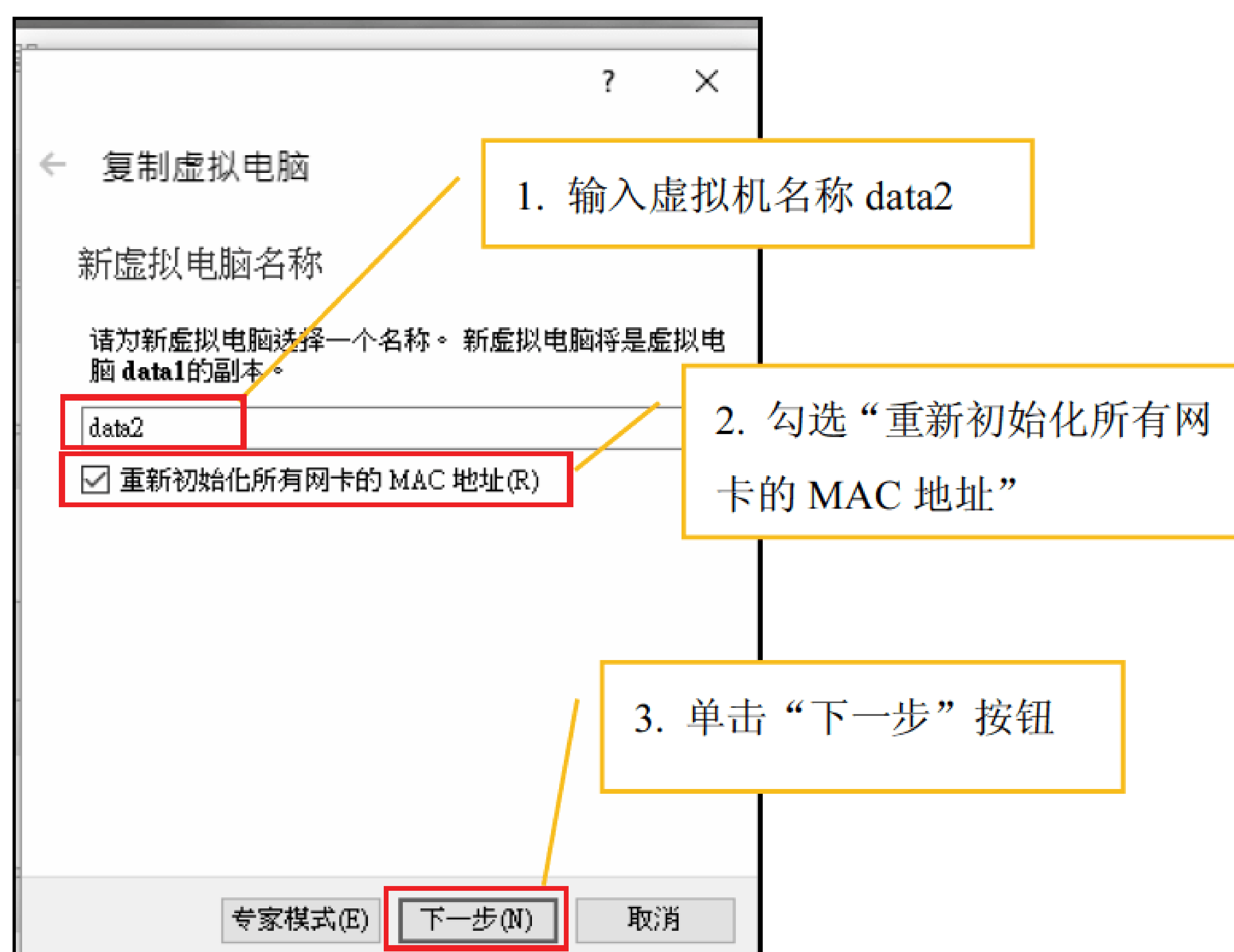


图 5-23　输入虚拟机名称

步骤 03 选择复制类型（见图 5-24）

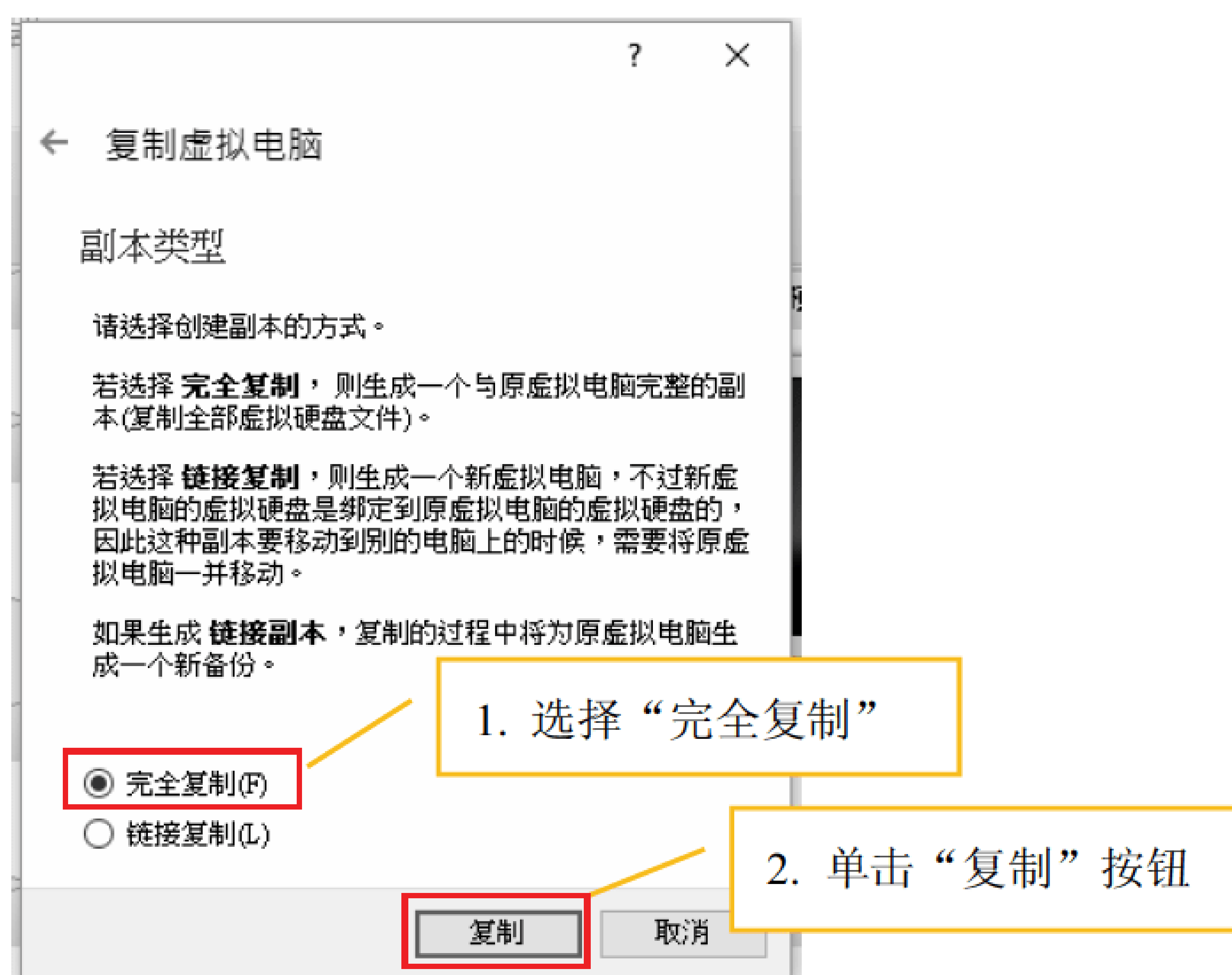


图 5-24　选择复制类型

步骤 04 data1 复制到 data2 虚拟机

单击"复制"按钮后，等候数分钟复制就完成了，接着会出现 data2 的图标，如图 5-25 所示。



图 5-25　复制完成

步骤 05　data1 复制到 data3、master 虚拟机

重复步骤 1~4，将 data1 虚拟机复制到 data3 虚拟机。

重复步骤 1~4，将 data1 虚拟机复制到 master 虚拟机。

如图 5-26 所示，我们已经将 data1 虚拟机复制到 data2、data3、master 虚拟机。



图 5-26　复制完成

步骤 06　选择虚拟机进行内存设置（见图 5-27）



图 5-27　选择虚拟机

步骤 07　虚拟机内存设置（见图 5-28）

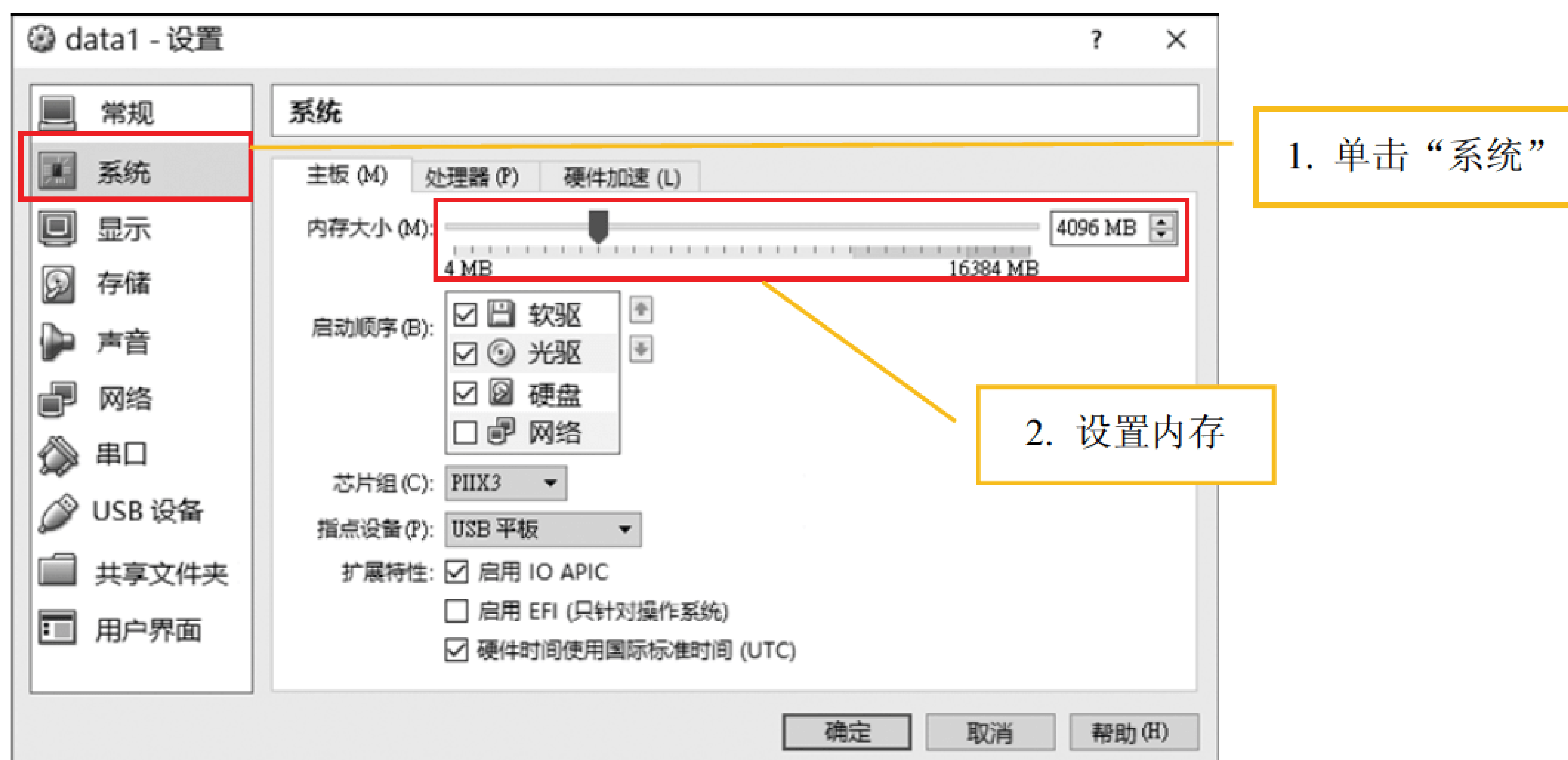


图 5-28　设置内存

关于虚拟机的内存设置，主要是由 Host 实体主机（PC 或服务器）的内存大小来决定：

- 如果物理内存是 16GB，建议设置为 master：4GB、data1：2GB、data2：2GB、data3：2GB。
- 如果物理内存是 8GB，建议设置为 master：2GB、data1：1GB、data2：1GB、data3：1GB。
- 如果物理内存只有 4GB，建议只使用 Single Node Cluster：2GB。

如果虚拟机的内存设置总和超过 Host 实体主机内存的大小，就可能会造成实体主机宕机。

5.5 设置 data2 服务器

接下来，设置 data2 的固定 IP、hostname。

步骤 01　启动 data2 虚拟机（见图 5-29）



图 5-29　启动 data2 虚拟机

步骤 02 设置 data2 固定 IP 地址

我们必须设置 data2 虚拟主机每次开机都是使用固定 IP 地址：192.168.56.102。在 data2 的“终端”程序中输入下列命令：

➢ 编辑 interfaces 网络配置文件

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-30 所示。

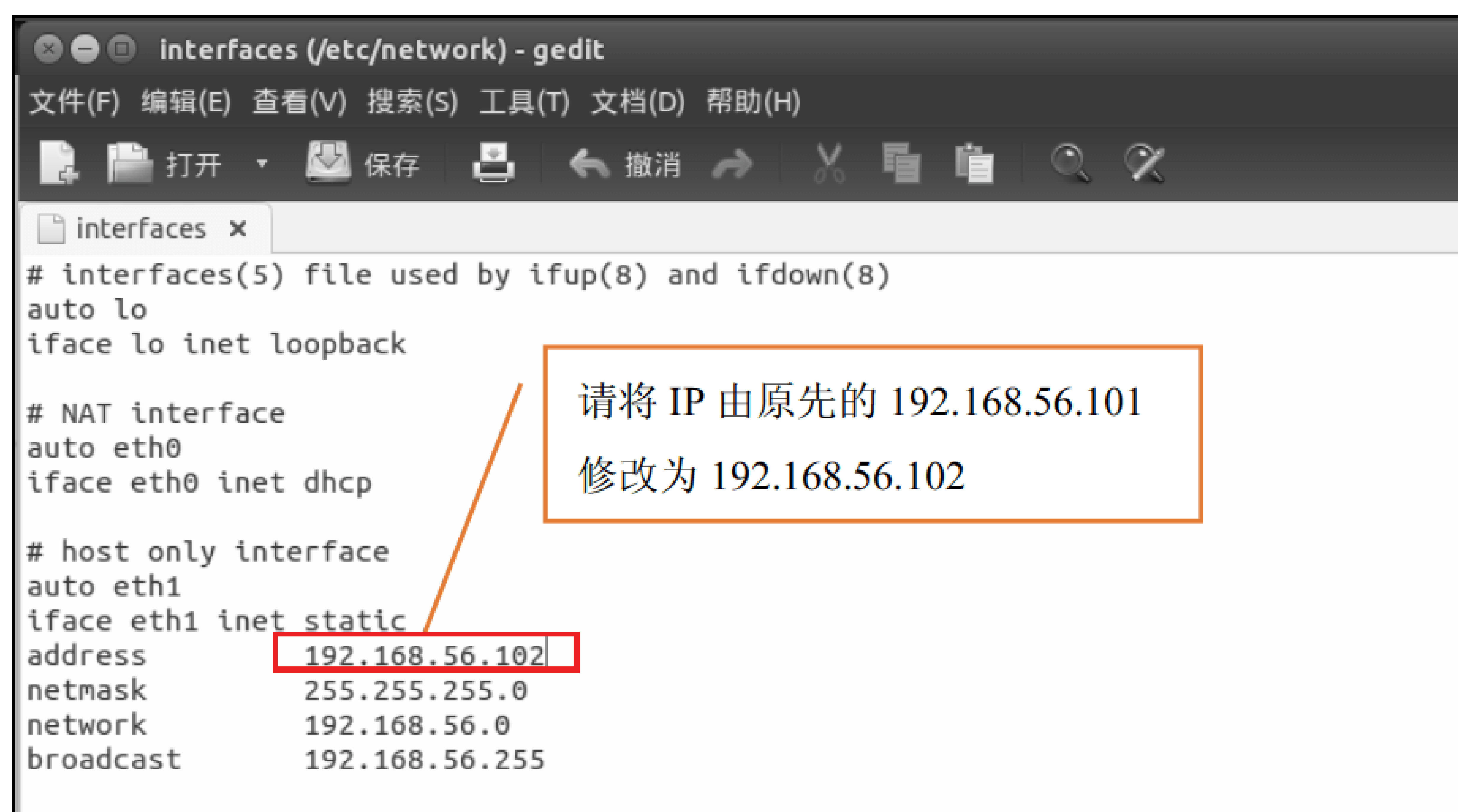


图 5-30　修改 IP 地址

编辑完成后，先保存，再关闭 gedit。

步骤 03　设置 data2 主机名

请在 data2 的“终端”程序中输入下列命令：

➢ **编辑 hostname 文件**

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-31 所示。



图 5-31　设置 data2 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04　重新启动 data2 虚拟机（见图 5-32）



图 5-32　重新启动

步骤 05　重新启动后确认 data2 虚拟机是否设置正确

重新启动后，在 data2“终端”程序中输入下列命令：

➢ **查看网络设置**

```
ifconfig
```

运行后结果如图 5-33 所示。从中可以看到提示符号已经改成 hduser@data2:~$，代表 hostname 已改为 data2，IP 已改为 192.168.56.102。

```
hduser@data2: ~
hduser@data2:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:66:e8:a6
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe66:e8a6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:90 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4549 (4.5 KB)  TX bytes:11384 (11.3 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:98:ef:63
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe98:ef63/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

1. hostname: data2

2. IP: 192.168.56.102

图 5-33　查看网络设置

5.6 设置 data3 服务器

接下来，设置 data3 的固定 IP、hostname。

步骤 01 启动 data3 虚拟机（见图 5-34）

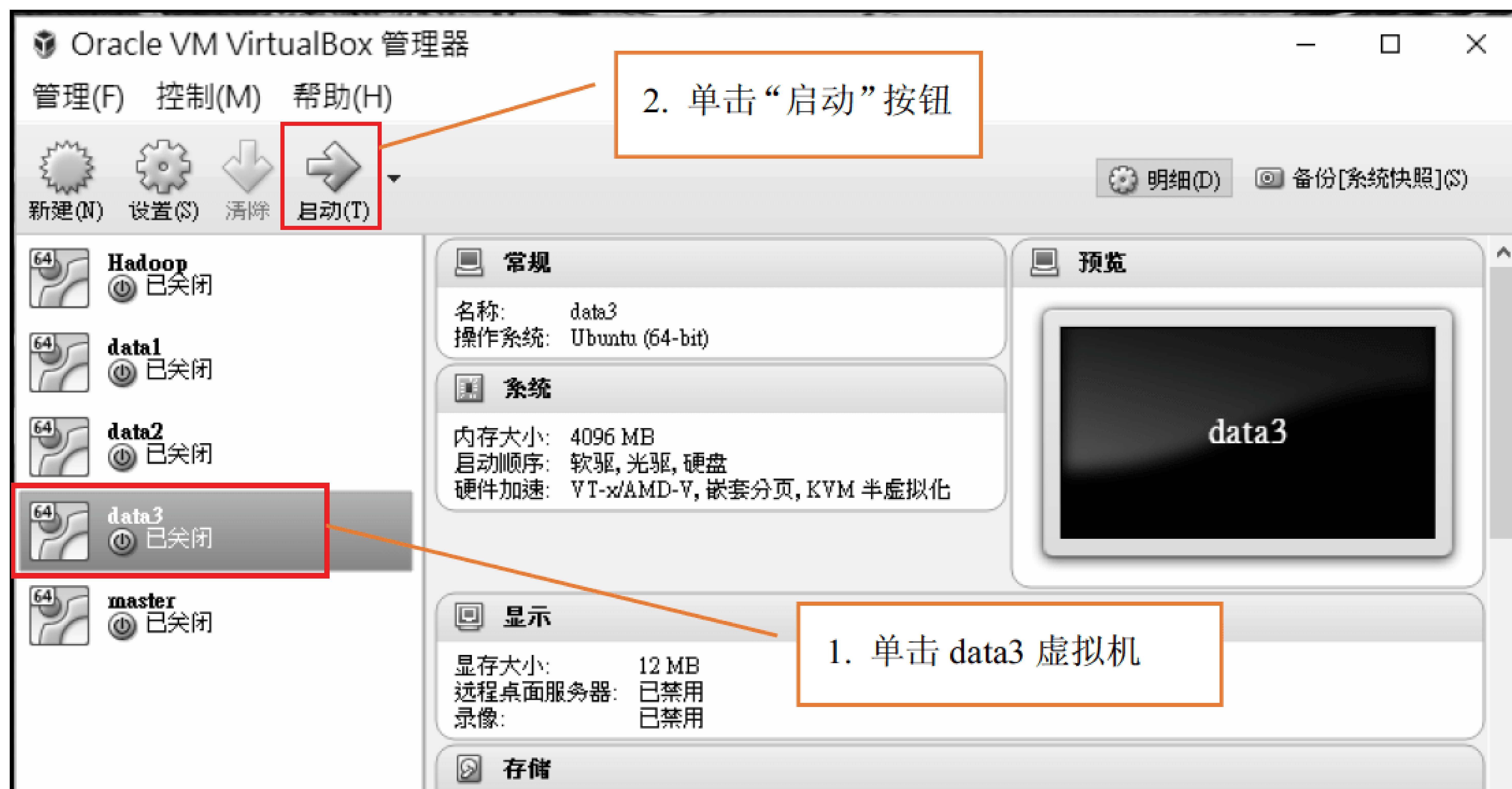


图 5-34　启动 data3 虚拟机

步骤 02 设置 data3 使用固定 IP 地址

接下来，我们要设置 data3 虚拟主机每次开机 IP 地址固定使用 192.168.56.103。

请在 data3 的“终端”程序中输入下列命令：

➢ **编辑 interfaces 网络配置文件**

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-35 所示。

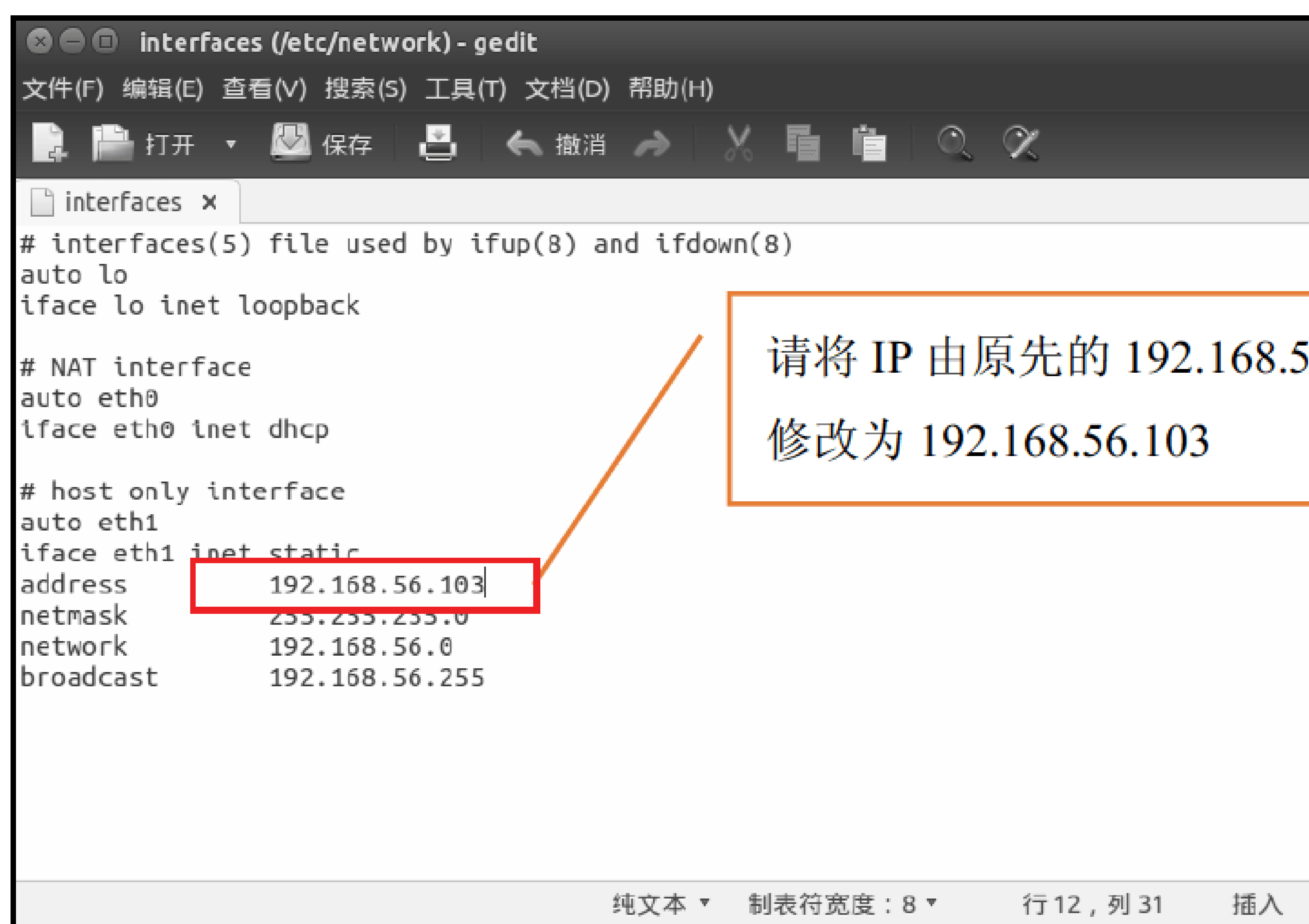


图 5-35　修改 IP

编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 data3 主机名

请在 data3 的“终端”程序中输入下列命令：

➢ **编辑 hostname 文件**

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-36 所示。



图 5-36　设置 data3 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04 重新启动 data3 虚拟机（见图 5-37）



图 5-37 重新启动

步骤 05 重新启动后确认 data3 是否正确（见图 5-38）

重新启动后，请在 data3 终端机中输入下列指令：

➤ **查看网络设置**

```
ifconfig
```

运行后结果如图 5-38 所示。从中可以看到提示符号已经改成 hduser@data3:~$，代表 hostname 已改为 data3，IP 已改为 192.168.56.103。

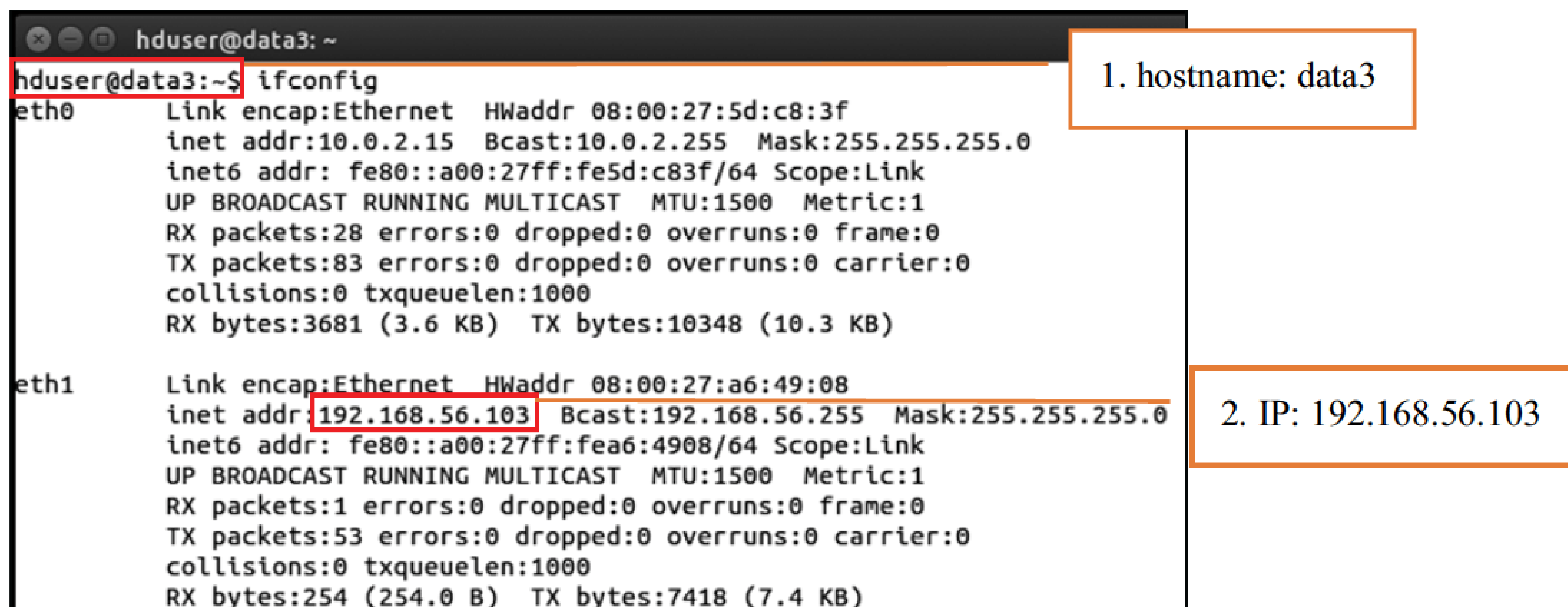


图 5-38 查看 data3 网络设置

5.7 设置 master 服务器

在 NameNode 服务器（master）中需要设置固定 IP 地址、hostname、hdfs-site.xml、masters、slaves。

步骤 01 启动 master 虚拟机（见图 5-39）



图 5-39　启动 master 虚拟机

步骤 02 设置 master 固定 IP 地址

我们必须设置 master 虚拟主机每次开机都使用固定 IP 地址 192.168.56.100。在 master 的“终端”程序中输入下列命令：

> **编辑 master 的 interfaces 网络配置文件**

```
sudo gedit /etc/network/interfaces
```

输入后按 Enter 键就会打开 interfaces 文件，屏幕显示界面如图 5-40 所示。



图 5-40　修改 IP

编辑完成后，先保存，再关闭 gedit。

步骤 03 设置 master 主机名

在 master 的“终端”程序中输入下列命令：

➢ **编辑 hostname 文件**

```
sudo gedit /etc/hostname
```

输入后按 Enter 键就会打开 hostname 文件，屏幕显示界面如图 5-41 所示。



图 5-41　设置 master 主机名

编辑完成后，先保存，再关闭 gedit。

步骤 04 设置 hdfs-site.xml

hdfs-site.xml 用于设置 HDFS 分布式文件系统相关配置的设置。因为 master 现在只是单纯的 NameNode，所以请删除 DataNode 的 HDFS 设置，并加入 NameNode 的 HDFS 设置。在 master 的“终端”程序中输入下列命令：

➢ **编辑 hdfs-site.xml**

```
sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

输入后按 Enter 键就会打开 hdfs-site.xml，屏幕显示界面如图 5-42 所示。

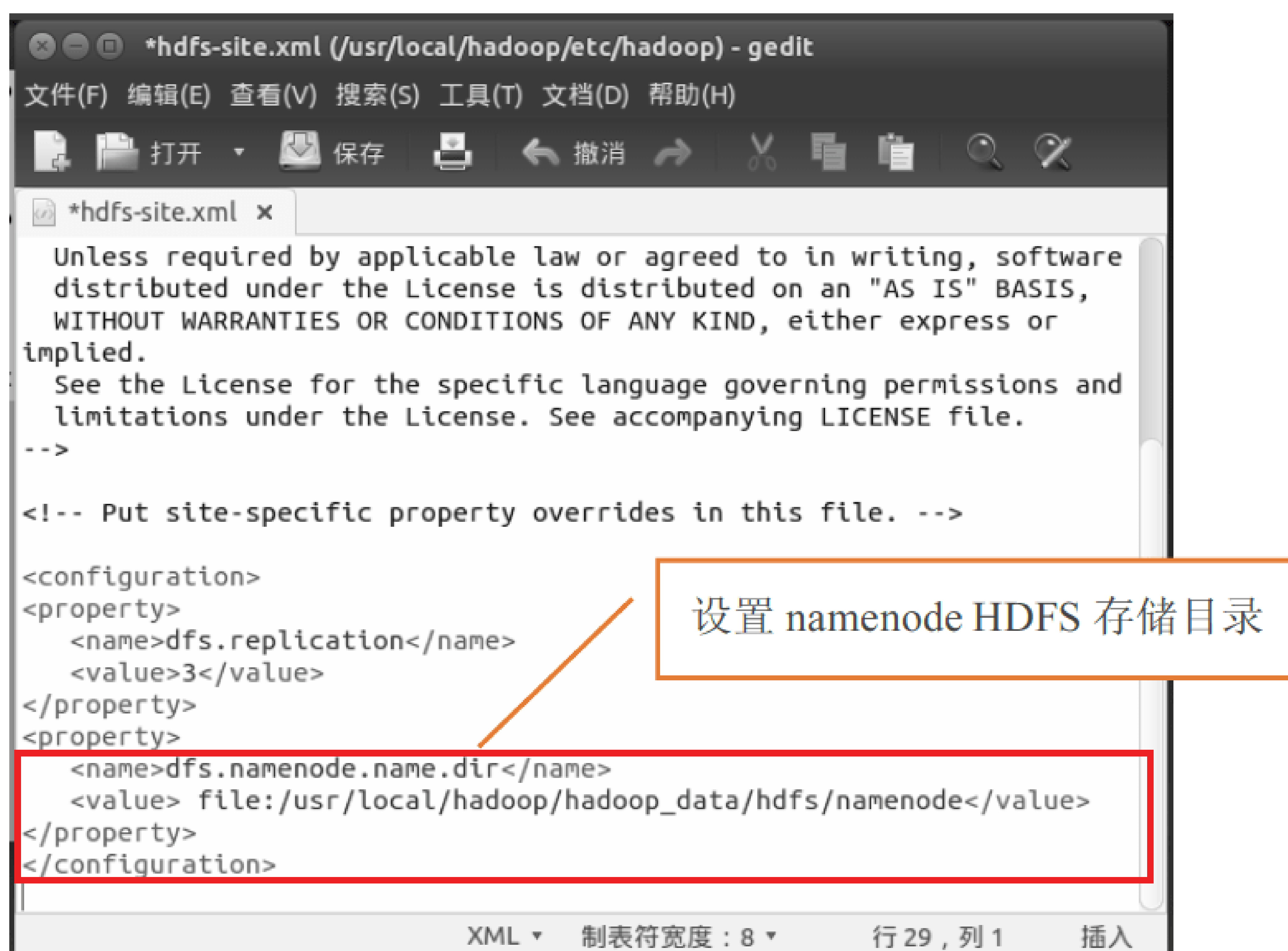


图 5-42　设置 hdfs-site.xml

编辑完成后，先保存，再关闭 gedit。

步骤 05 编辑 masters 文件

masters 文件主要是告诉 Hadoop 系统哪一台服务器是 NameNode。
在 master 的“终端”程序中输入下列命令：

➢ **编辑 master 文件**

```
sudo gedit /usr/local/hadoop/etc/hadoop/masters
```

输入后按 Enter 键就会打开 master，屏幕显示界面如图 5-43 所示。



图 5-43　编辑 master 文件

编辑完成后，先保存，再关闭 gedit。

步骤 06 编辑 slaves 文件

slaves 文件主要是告诉 Hadoop 系统哪些服务器是 DataNode。
在 master 的“终端”程序中输入下列命令：

➢ **编辑 slaves 文件**

```
sudo gedit /usr/local/hadoop/etc/hadoop/slaves
```

输入后按 Enter 键就会打开 master，屏幕显示界面如图 5-44 所示。



图 5-44　编辑 slaves 文件

编辑完成后，先保存，再关闭 gedit。

步骤 07 重新启动 master 虚拟机（见图 5-45）



图 5-45 重新启动

步骤 08 重新启动后确认 master 虚拟机是否设置正确（见图 5-46）

重新启动后，请在 master 终端机输入下列指令：

➢ **查看网络设置**

```
ifconfig
```

运行后如图 5-46 所示，可以看到提示符号已经改成 hduser@master:~$，代表 hostname 已改为 master，IP 也已改为 192.168.56.100。

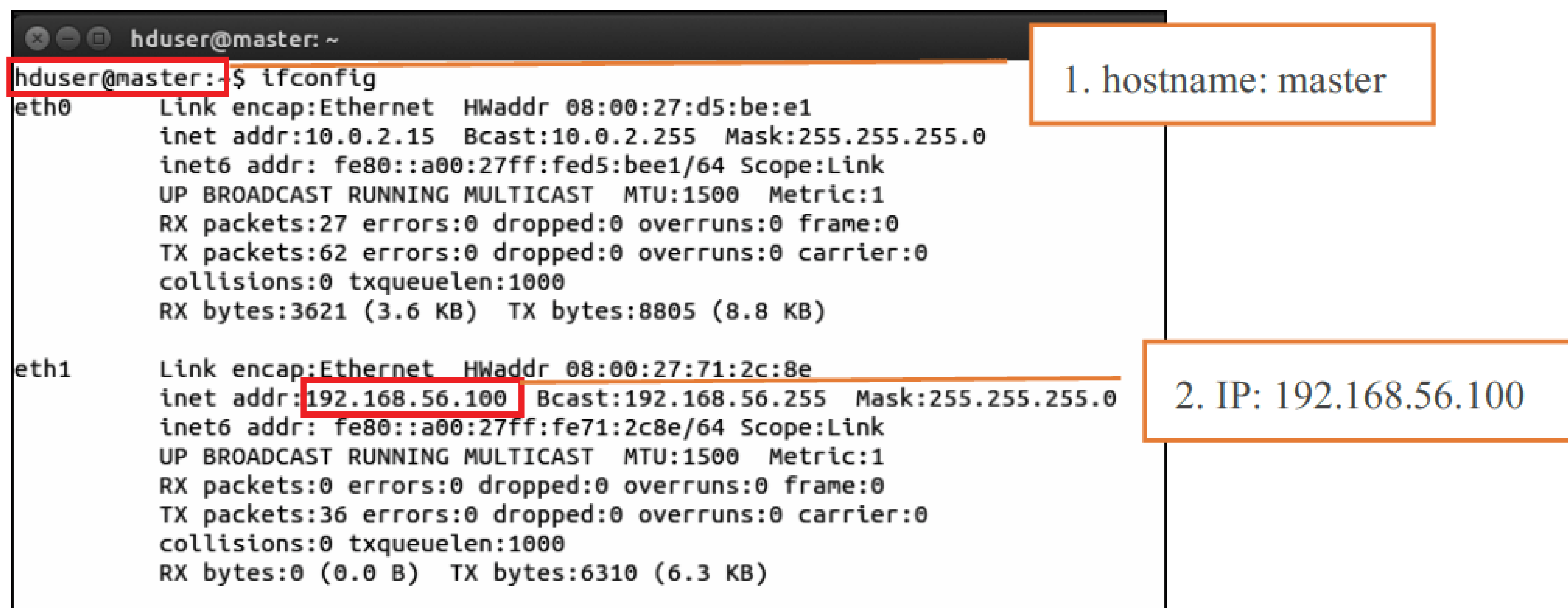


图 5-46 查看 master 网络设置

5.8 master 连接到 data1、data2、data3 创建 HDFS 目录

之前的步骤我们已经创建了 master 与 data1、data2、data3 服务器。接下来，创建 NameNode（master）的 SSH 连接到 DataNode（data1、data2、data3），并创建 HDFS 相关目录。

步骤 01 启动 master、data1、data2、data3

首先，必须要启动所有虚拟服务器 master、data1、data2、data3，如图 5-47 所示。



图 5-47　启动所有虚拟服务器

切换到 master 虚拟机之后启动"终端"程序。我们将以 master "终端" 程序通过 SSH 连接到 data1 服务器。

步骤 02 连接到 data1 虚拟机

在 master 的"终端"程序中输入下列命令：

➢ **master 通过 SSH 连接到 data1 虚拟机**

```
ssh data1
```

如图 5-48 所示，输入命令后按 Enter 键，就会连接到 data1。

请注意！提示符原本是 hduser@master:~$，连接到 data1 之后变成了 hduser@data1:~$，代表已经成功连接 data1。

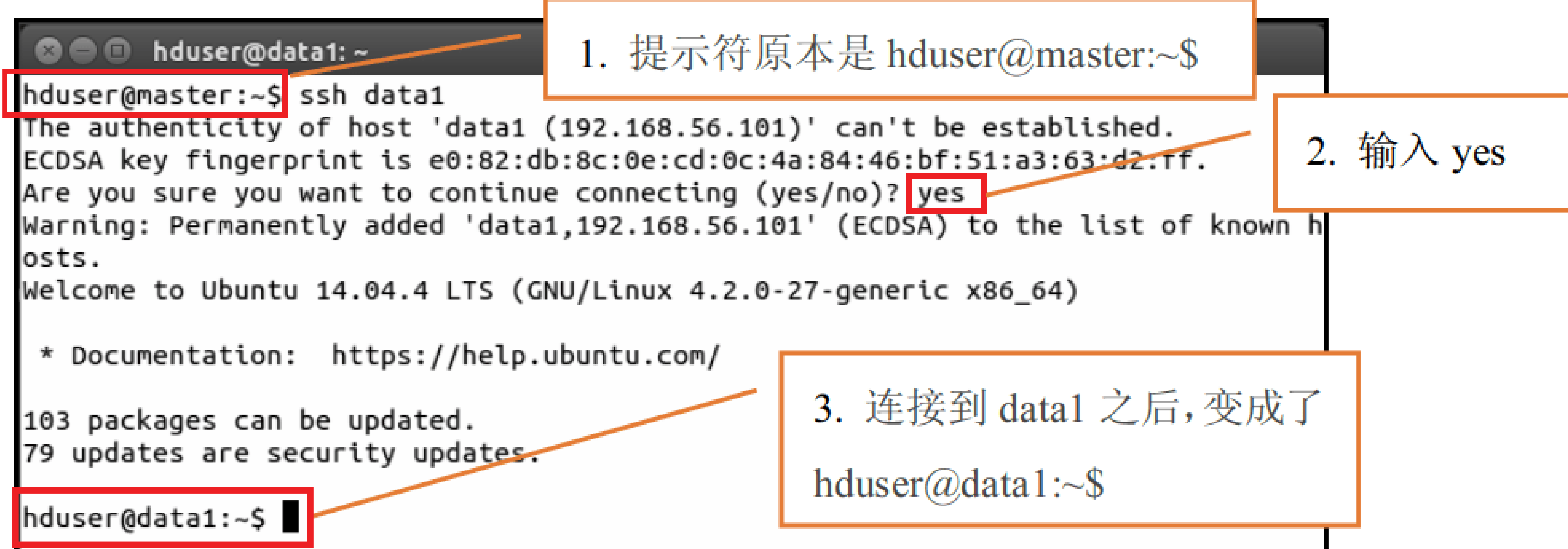


图 5-48　连接到 data1 虚拟机

步骤 03 连接到 data1 创建 HDFS 相关目录

登录 data1 后，我们将在 data1 创建 HDFS 相关目录。

在 master 的“终端”程序中输入下列命令：

➢ **删除 HDFS 所有目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➢ **创建 DataNode 存储目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➢ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

完成后，屏幕显示界面如图 5-49 所示。

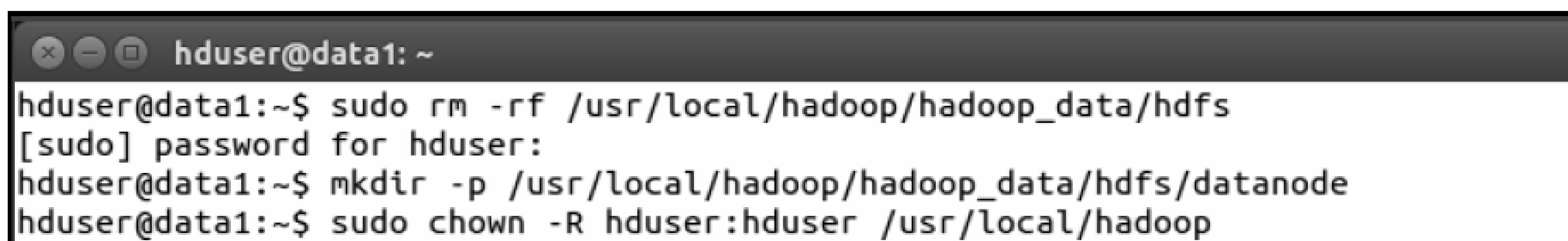


图 5-49　创建 HDFS 相关目录

步骤 04 中断 data1 连接，回到 master

完成后，在 master 的“终端”中输入下列命令：

➢ **中断 data1 连接，回到 master**

```
exit
```

如图 5-50 所示，运行 exit 之后，原本提示符是 hduser@data1:~$，中断 data1 连接后恢复成 hduser@master:~$，代表连接中断回到 master 虚拟机。

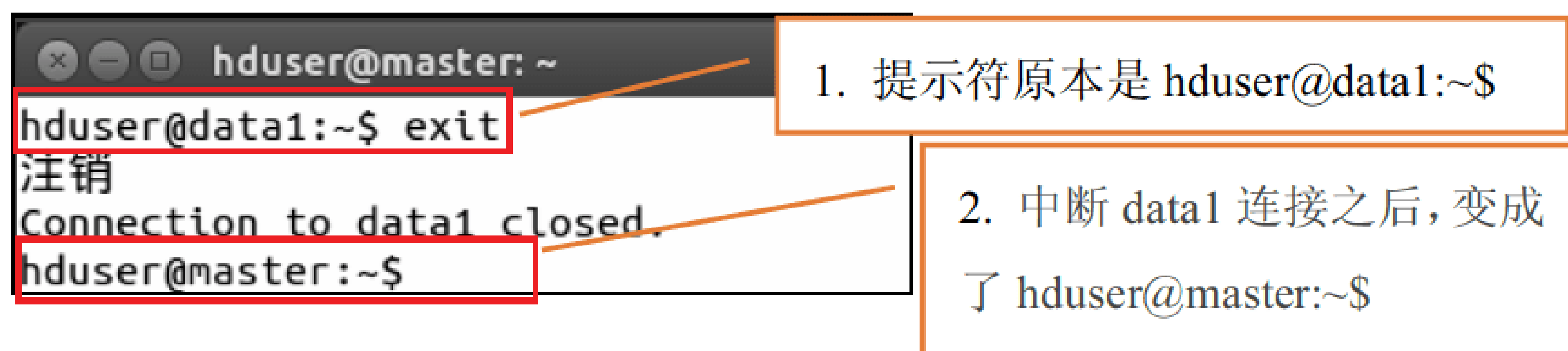


图 5-50　回到 master

步骤 05 master SSH 连接到 data2 并创建 HDFS 目录

接下来，参照上一小节相同的做法，使用下列命令让 master 通过 SSH 连接到 data2 虚拟机，并且在 data2 虚拟机创建 HDFS 目录。

在 master“终端”程序中输入下列命令：

➢ **master 通过 SSH 连接到 data2 虚拟机**

```
ssh data2
```

➢ **删除 HDFS 所有目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➢ **创建 DataNode 存储目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➢ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

➢ **中断 data2 连接，回到 master**

```
exit
```

完成后屏幕显示界面如图 5-51 所示。

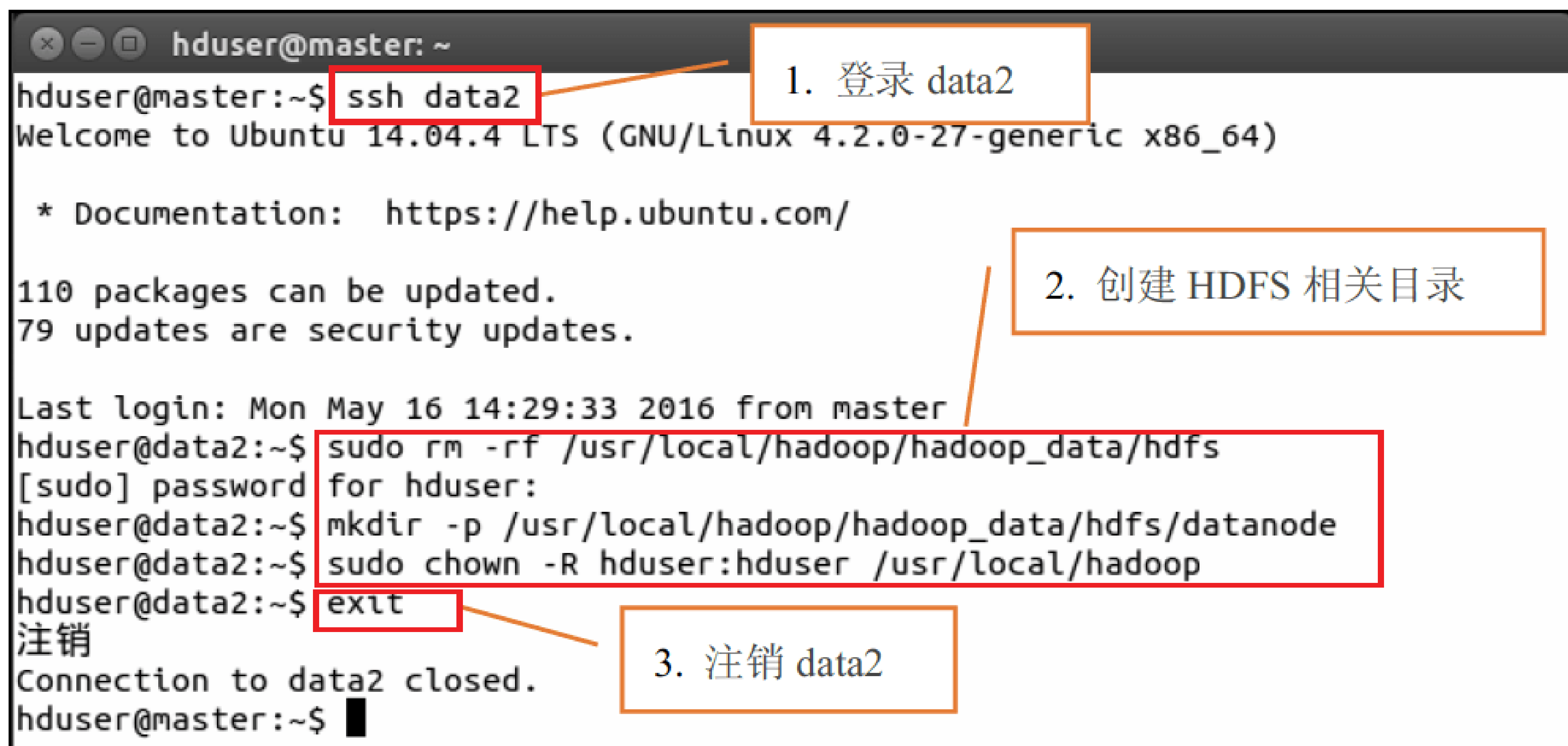


图 5-51　master SSH 连接到 data2 并创建 HDFS 目录

步骤06 master SSH 连接到 data3 并创建 HDFS 目录

参照相同的方法，我们使用下列命令让 master 通过 SSH 连接到 data3 虚拟机，并且在 data3 虚拟机创建 HDFS 目录。

在 master 的“终端”程序中输入下列命令：

➢ **master 通过 SSH 连接到 data3 虚拟机**

```
ssh data3
```

➢ **删除 HDFS 所有目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➢ **创建 DataNode 存储目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
```

➢ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

➢ **中断 data3 连接，回到 master**

```
exit
```

完成后屏幕显示界面如图 5-52 所示。

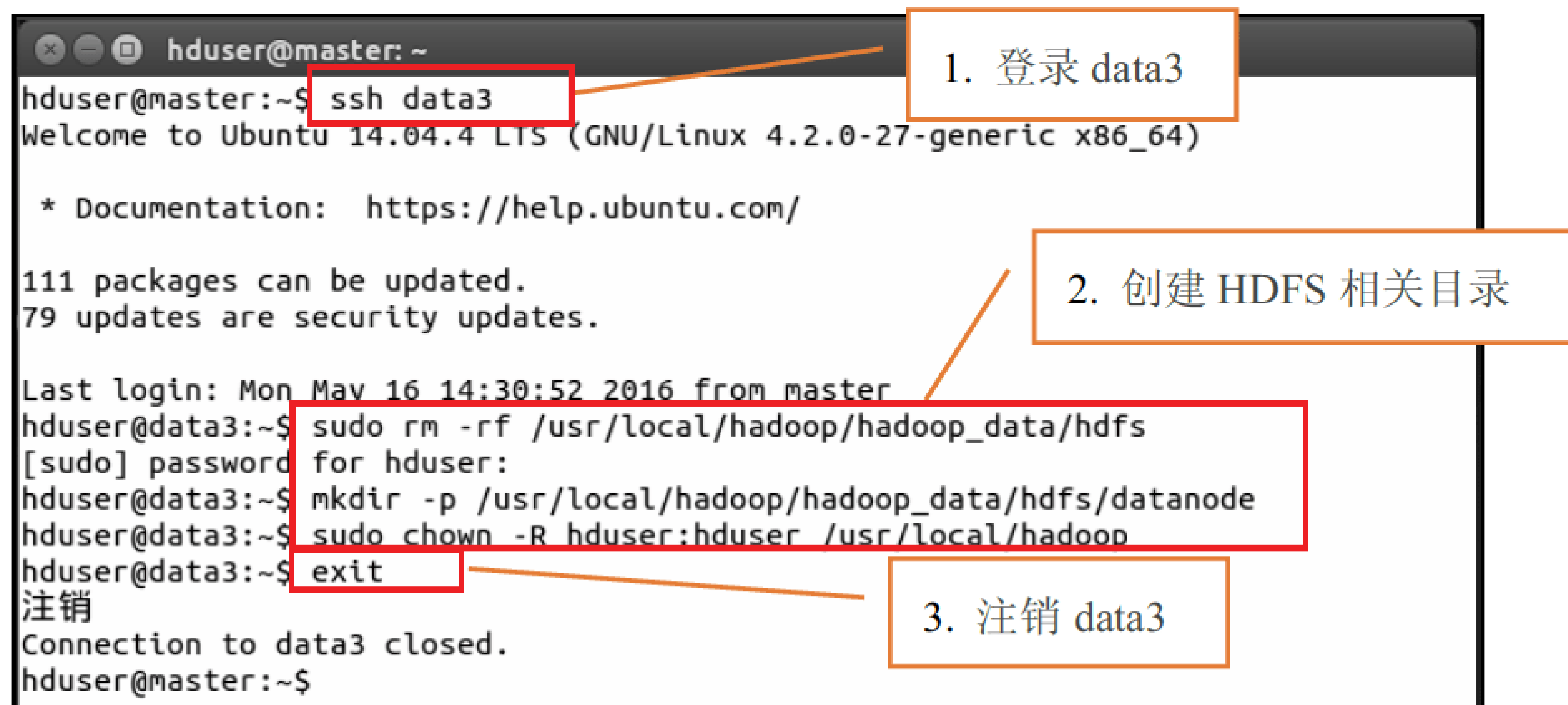


图 5-52 master SSH 连接到 data3 并创建 HDFS 目录

5.9 创建并格式化 NameNode HDFS 目录

步骤01 重新创建 NameNode HDFS 目录

在 master 的“终端”程序中输入下列命令，创建 NameNode HDFS 目录：

➢ **删除之前的 HDFS 目录**

```
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
```

➢ **创建 NameNode 目录**

```
mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
```

➢ **将目录的所有者更改为 hduser**

```
sudo chown -R hduser:hduser /usr/local/hadoop
```

在 master 虚拟机的“终端”程序中输入如图 5-53 所示的命令。

```
hduser@master: ~
hduser@master:~$ sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs
hduser@master:~$ mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
hduser@master:~$ sudo chown -R hduser:hduser /usr/local/hadoop
hduser@master:~$
```

图 5-53　输入命令

步骤 02　格式化 NameNode HDFS 目录

之前我们已经创建了 DataNode 与 NameNode 的 HDFS 目录，接下来，需要格式化 HDFS 目录。在 master 的“终端”程序中输入下列命令：

➢ **格式化 NameNode HDFS 目录**

```
hadoop namenode -format
```

格式化时屏幕显示界面如图 5-54 所示。

```
hduser@master: ~
hduser@master:~$ hadoop namenode -format
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

15/04/29 11:07:03 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = master/192.168.0.100
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.6.0
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/
hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/jac
kson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/
usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/s
hare/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/usr/local/hadoop/share/hadoo
p/common/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore
-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/jasper-compiler-5.5.23.jar:
/usr/local/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/usr/local/had
oop/share/hadoop/common/lib/jasper-runtime-5.5.23.jar:/usr/local/hadoop/share/ha
doop/common/lib/curator-framework-2.6.0.jar:/usr/local/hadoop/share/hadoop/commo
n/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.j
```

图 5-54　格式化 NameNode HDFS 目录

5.10 启动 Hadoop Multi Node Cluster

步骤 01 启动 Hadoop Multi Node Cluster

到目前为止我们已经完成 Hadoop cluster 的构建，可以在 master 的“终端”程序中输入下列命令开始操作：

➢ **分别启动 HDFS 与 YARN**

命令	说明
start-dfs.sh	启动 HDFS
start-YARN.sh	启动 Hadoop MapReduce 框架 YARN

或

➢ **同时启动 HDFS 与 YARN**

命令	说明
start-all.sh	同时启动 HDFS 与 YARN

如图 5-55 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
The authenticity of host 'master (192.168.56.100)' can't be established.
ECDSA key fingerprint is 67:00:38:8d:57:5e:64:58:d5:6a:92:8c:c7:65:22:8f.
Are you sure you want to continue connecting (yes/no)? yes
master: Warning: Permanently added 'master,192.168.56.100' (ECDSA) to the list o
f known hosts.
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namen
ode-master.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data3.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data2.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-master.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data3.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data1.out
hduser@master:~$
```

1. 第1次执行会询问，请输入 yes

2. 启动 HDFS

3. 启动 YARN

图 5-55 启动 Hadoop Multi Node Cluster

步骤02 查看 master（NameNode）进程（process）

jps（Java Virtual Machine Process Status Tool）可以用于查看当前所运行的进程，在 master 的“终端”程序中输入下列命令：

➢ **查看 master（NameNode）的进程（process）是否启动**

```
jps
```

运行后屏幕显示界面如图 5-56 所示，可以看见 master 服务器的状态。

- **HDFS 功能：** NameNode、SecondaryNameNode 已经启动。
- **MapReduce2（YARN）：** ResourceManager 已经启动。



图 5-56　查看启动的进程

步骤03 查看 data1（DataNode）进程

在 master 的“终端”程序中输入下列命令，通过 SSH 连接到 data1 来查看 data1（DataNode）进程。

➢ **master 通过 SSH 连接到 data1 虚拟机**

```
ssh data1
```

➢ **查看 data1（DataNode）所运行的进程**

```
jps
```

➢ **注销 data1 回到 master**

```
exit
```

运行后屏幕显示界面如图 5-57 所示，从中可以看见 data1 服务器的状态。

- **HDFS 功能：** DataNode 已经启动。
- **MapReduce2（YARN）：** NodeManager 已经启动。

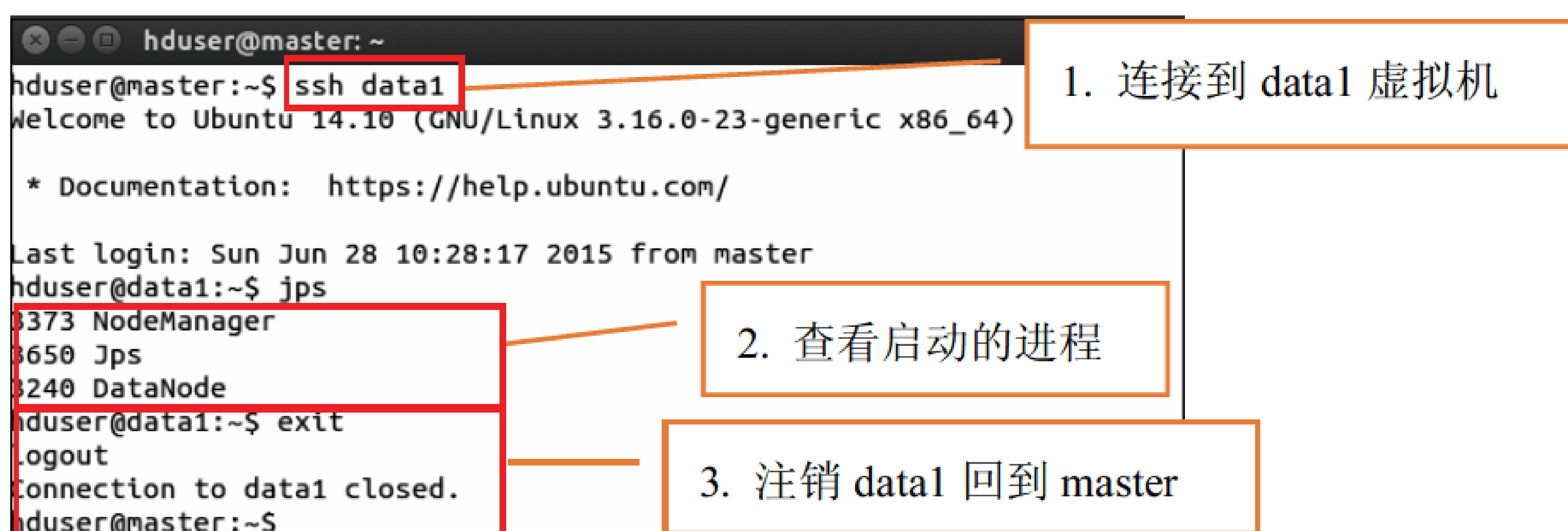


图 5-57 查看 data1 进程

也可以使用相同方式连接到 data2、data3 进程。

5.11 打开 Hadoop ResourceManager Web 界面

Hadoop ResourceManager Web 界面可用于查看当前 Hadoop 的状态：Node 节点、应用程序、进程运行情况。

步骤 01 打开 ResourceManager Web 界面

打开浏览器网址栏输入：

➢ **ResourceManager Web 界面网址**

```
http://master:8088/
```

参照下列步骤，就可以看到如图 5-58 所示的屏幕显示界面。

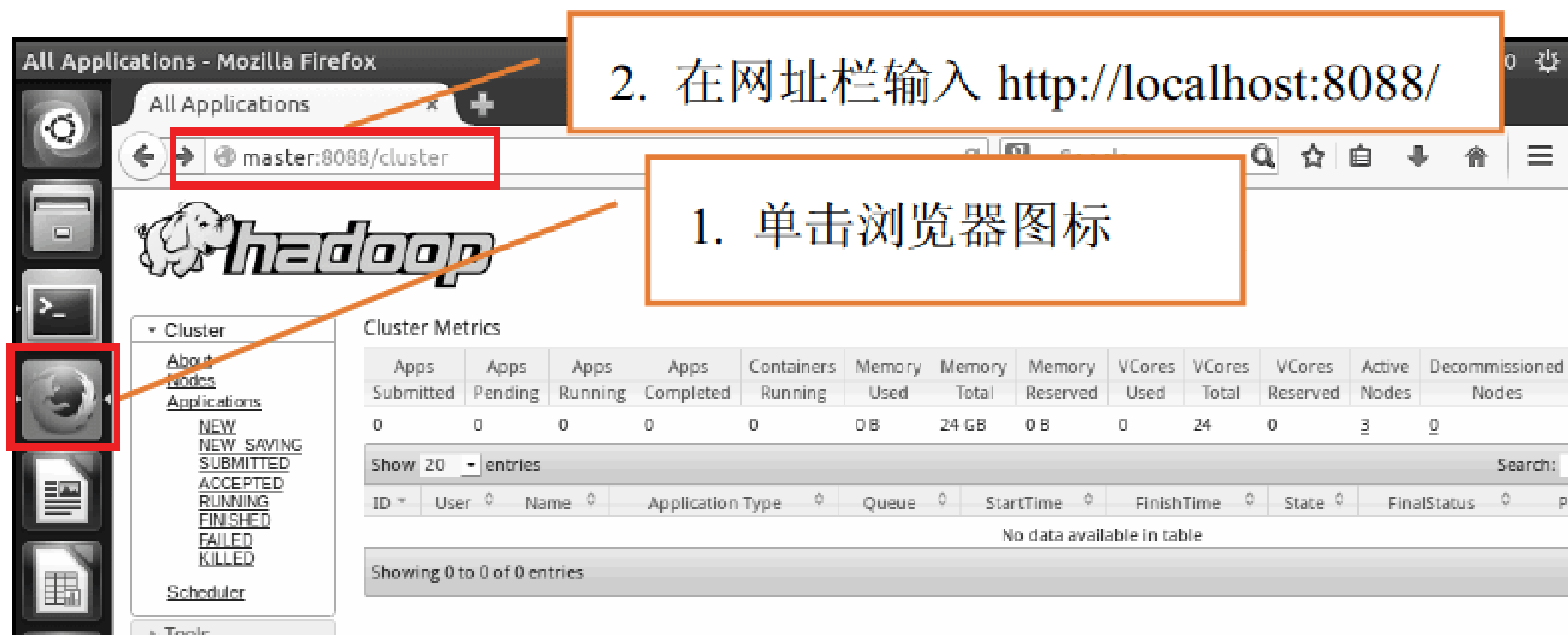


图 5-58 打开 Resource Manager Web 界面

步骤 02 查看已经运行的节点 Nodes

当单击 Nodes 时，会显示当前的节点。当前共有 3 个节点 Nodes：data1、data2、data3，

如图 5-59 所示（单击 Nodes 时，有时没有出现节点，刷新几次就会出现）。

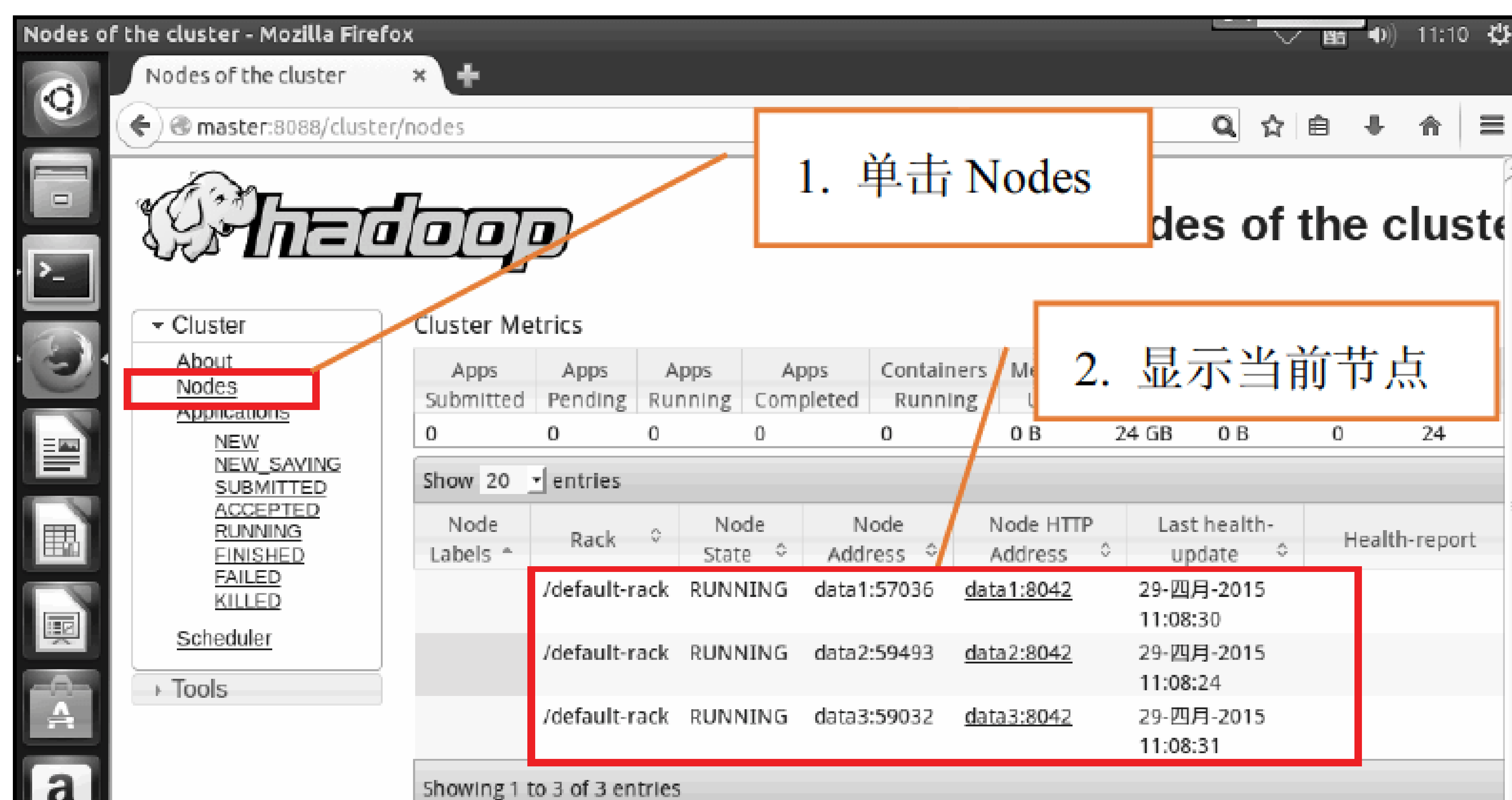


图 5-59　查看已经运行的节点

5.12　打开 NameNode Web 界面

HDFS Web 界面可用于检查当前的 HDFS 与 DataNode 运行情况。

步骤 01 打开 NameNode HDFS Web 用户界面

打开浏览器在网址栏输入：

➢ **HDFS Web 用户界面网址**

```
http://master:50070/
```

可以看到屏幕显示界面如图 5-60 所示。

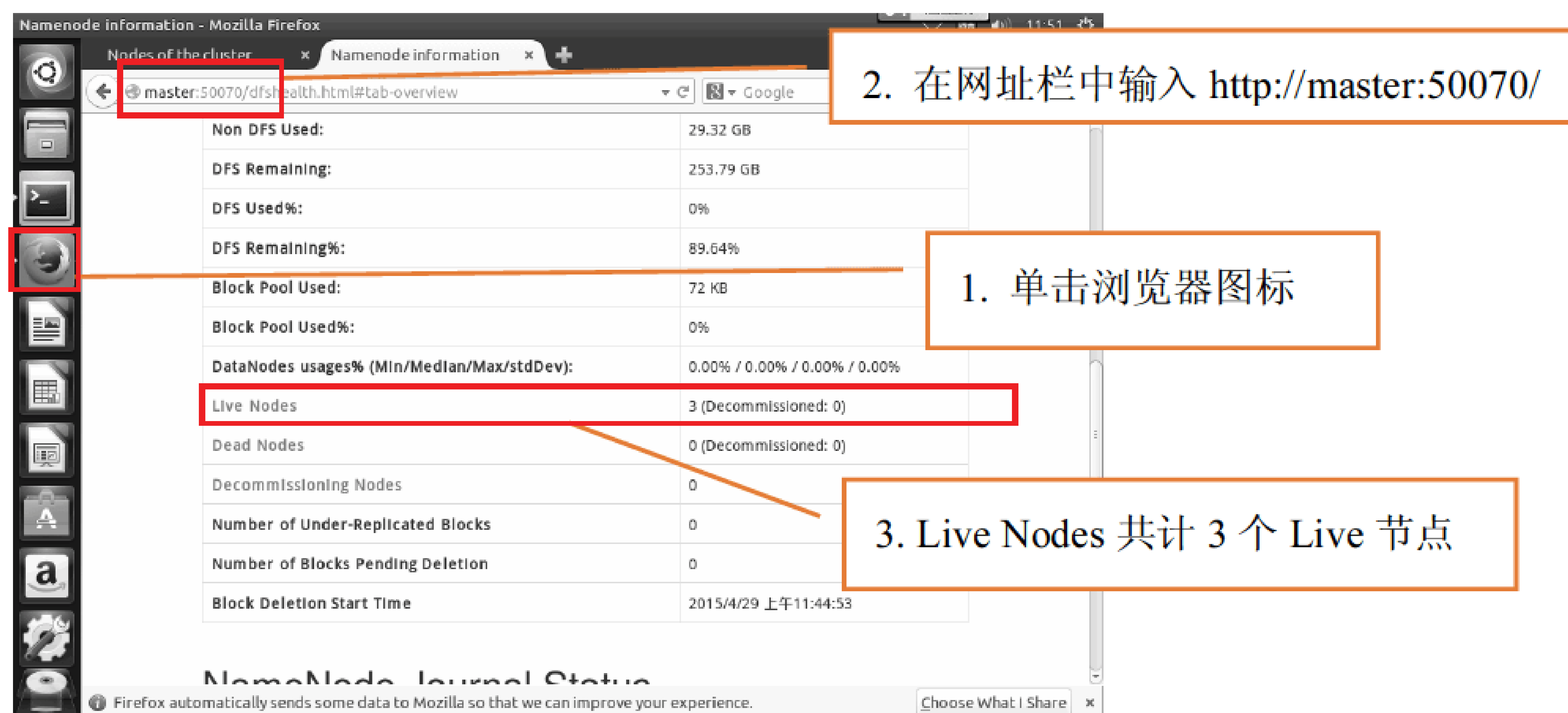


图 5-60　输入网址

步骤 02 查看 Datanode

单击 Datanodes，可以看到当前启动了 3 个节点 Datanodes，如图 5-61 所示。

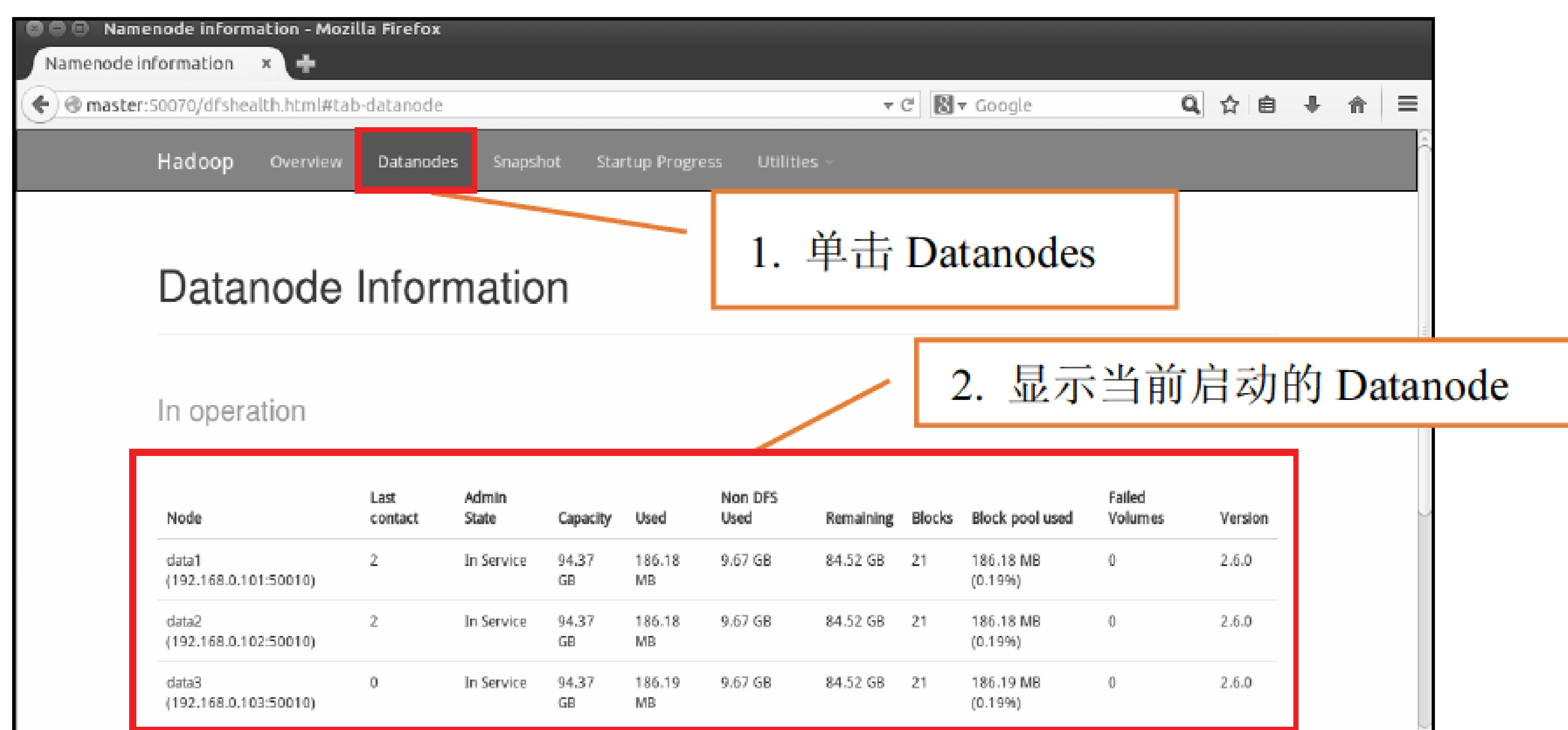


图 5-61 查看 Datanode

5.13 停止 Hadoop Multi Node Cluster

在 master “终端”程序中输入下列命令：

➢ **停止** Hadoop Multi Node Cluster（**见图** 5-62）

```
stop-all.sh
```

```
hduser@master: ~
hduser@master:~$ stop-all.sh
This script is Deprecated. Instead use stop-dfs.sh and stop-yarn.sh
Stopping namenodes on [master]
master: stopping namenode
data2: stopping datanode
data1: stopping datanode
data3: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
stopping yarn daemons
stopping resourcemanager
data2: stopping nodemanager
data3: stopping nodemanager
data1: stopping nodemanager
no proxyserver to stop
```

图 5-62 停止 Hadoop Nulti Node Cluster

5.14 结论

本章我们介绍了如何建立 Hadoop Multi Node Cluster。通过安装设置过程，相信读者已经对 Hadoop 有了基本的概念。下一章，我们将介绍 Hadoop HDFS 分布式存储。

第 6 章 Hadoop HDFS命令

在第 1 章中我们介绍了 HDFS 的概念，在本章中我们将介绍可以在“终端”程序中使用的 HDFS 命令，对 HDFS 进行操作，以及 Hadoop HDFS Web 接口。

利用终端 HDFS 命令对 HDFS 进行操作的示意图如图 6-1 所示。

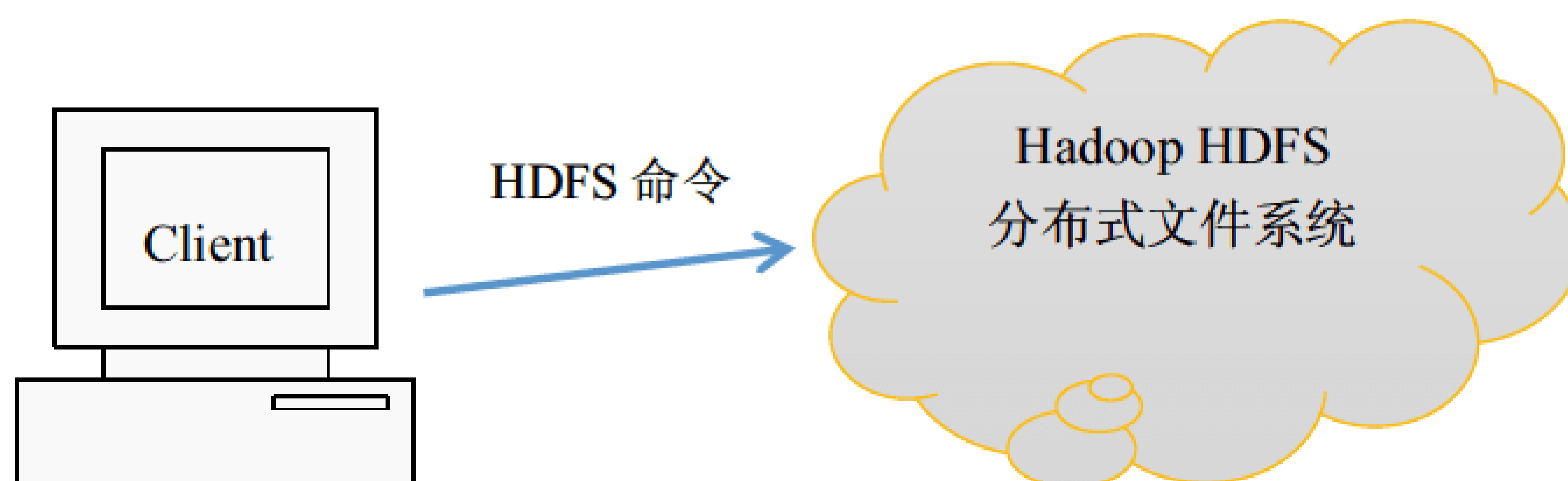


图 6-1　利用 HDFS 命令对 HDFS 进行操作

HDFS 命令格式如下：

```
Hadoop fs -命令
```

➢ **本章介绍一些常用的 HDFS 命令（见表 6-1）**

本章所有命令，我们都在 master 虚拟机的“终端”程序中运行。

表 6-1　常用的 HDFS 命令

命令	说明
hadoop fs -mkdir	创建 HDFS 目录
hadoop fs -ls	列出 HDFS 目录
hadoop fs -copyFromLocal	使用-copyFromLocal 复制本地（local）文件到 HDFS
hadoop fs -put	使用-put 复制本地（local）文件到 HDFS
hadoop fs -cat	列出 HDFS 目录下的文件内容
hadoop fs -copyToLocal	使用-copyToLocal 将 HDFS 上的文件复制到本地（local）
hadoop fs -get	使用-get 将 HDFS 上的文件复制到本地（local）
hadoop fs -cp	复制 HDFS 文件
hadoop fs -rm	删除 HDFS 文件

➢ **HDFS 命令整理**

常用的 HDFS 命令已被整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可节省打字的时间，也不用担心打错字（无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

6.1 启动 Hadoop Multi-Node Cluster

在示范 HDFS 命令之前，我们必须先启动 Hadoop Multi-Node Cluster。

步骤 01 启动所有虚拟服务器

首先必须要启动所有虚拟服务器 master、data1、data2、data3，如图 6-2 所示。



图 6-2　启动所有虚拟服务器

步骤 02 进入 master 虚拟机，启动 Hadoop Multi-Node Cluster

在 master 虚拟机启动“终端”程序，输入下列命令：

➢ **启动 Hadoop Multi-Node Cluster**

```
start-all.sh
```

运行后屏幕显示界面如图 6-3 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namen
ode-master.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data2.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-master.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data3.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodem
anager-data1.out
hduser@master:~$
```

图 6-3　启动 Hadoop Multi-Node Cluster

6.2 创建与查看 HDFS 目录

步骤01 创建 HDFS 目录

在“终端”程序中输入下列命令，创建 HDFS 目录：

➢ **创建 user 目录**

```
hadoop fs -mkdir /user
```

➢ **在 user 目录下创建 hduser 子目录**

```
hadoop fs -mkdir /user/hduser
```

➢ **在 hduser 目录下创建 test 子目录**

```
hadoop fs -mkdir /user/hduser/test
```

运行后屏幕显示界面如图 6-4 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -mkdir /user
hduser@master:~$ hadoop fs -mkdir /user/hduser
hduser@master:~$ hadoop fs -mkdir /user/hduser/test
hduser@master:~$
```

图 6-4 创建 HDFS 目录

步骤02 查看用户 HDFS 目录

在“终端”程序中输入下列命令：

➢ **查看之前创建的 HDFS 目录**

```
hadoop fs -ls
```

因为当前登录的用户是 hduser，所以会显示/user/hduser 下的目录，也就是 test 目录，如图 6-5 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -ls
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2015-05-08 21:12 test
```

图 6-5 查看 HDFS 目录

步骤03 查看 HDFS 完整目录

在“终端”程序中输入下列命令，查看之前创建的完整 HDFS 目录。因为 hadoop fs -ls 只能查看一级目录，所以必须逐级查看，如下列范例所示。

➢ **查看 HDFS 根目录**

```
hadoop fs -ls /
```

➢ **查看 HDFS 的/user 目录**

```
hadoop fs -ls /user
```

➢ **查看 HDFS 的/user/hduser 目录**

```
hadoop fs -ls /user/hduser
```

运行后屏幕显示界面如图 6-6 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -ls /
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user
hduser@master:~$ hadoop fs -ls /user
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser
hduser@master:~$ hadoop fs -ls /user/hduser
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser/test
hduser@master:~$
```

图 6-6　逐级查看目录

步骤 04　查看所有 HDFS 子目录

HDFS 提供了一个方便的选项，我们可以加上-R，R 代表 recursive（递归），可进行如下操作：

➢ **一次查看所有子目录**

```
hadoop fs -ls -R /
```

运行后屏幕显示界面如图 6-7 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -ls -R /
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser/test
hduser@master:~$
```


图 6-7　一次查看所有子目录

步骤 05　一次创建所有 HDFS 子目录

当我们创建目录时，如果要逐级地创建也很麻烦，所以 HDFS 提供了-p 选项，可以帮助用户一次创建多级目录。在“终端”程序中输入下列命令：

➢ **创建多级 HDFS 目录**

```
hadoop fs -mkdir -p /dir1/dir2/dir3
```

➢ **查看所有 HDFS 子目录**

```
hadoop fs -ls -R /
```

运行后屏幕显示界面如图 6-8 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -mkdir -p /dir1/dir2/dir3
hduser@master:~$ hadoop fs -ls -R /
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:54 /dir1
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:54 /dir1/dir2
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:54 /dir1/dir2/dir3
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser
drwxr-xr-x   - hduser supergroup          0 2015-05-08 09:51 /user/hduser/test
hduser@master:~$
```

系统帮助创建多层目录

图 6-8　一次创建多层目录

6.3 从本地计算机复制文件到 HDFS

从本地计算机复制文件到 HDFS 的示意图如图 6-9 所示。

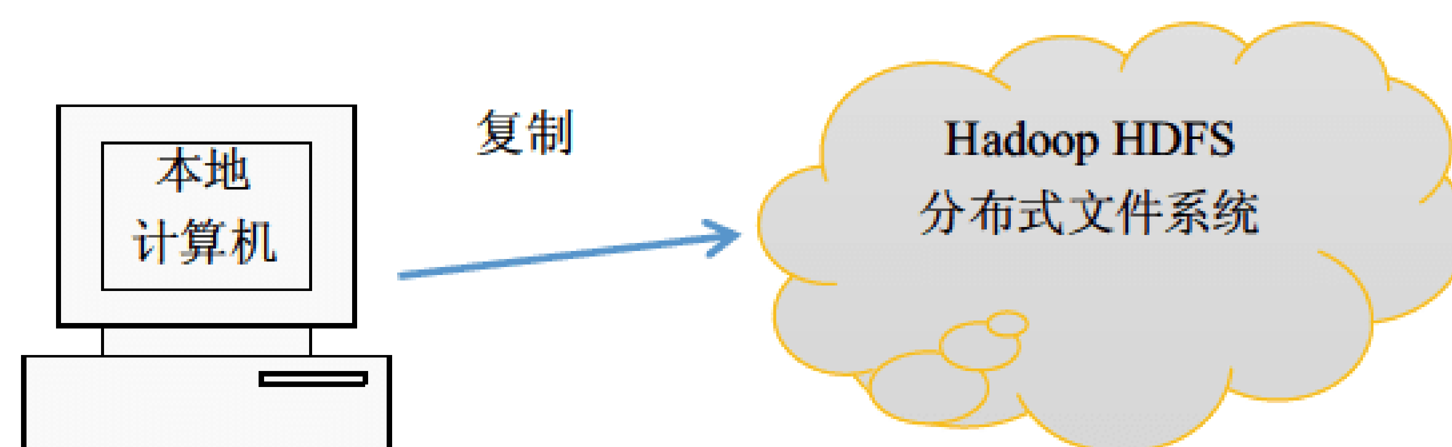


图 6-9　从本地计算机复制文件到 HDFS 示意图

步骤 01 复制本地（local）文件到 HDFS

使用下列命令复制本地（local）文件到 HDFS：

➢ **复制本地文件到 HDFS 的目录**

```
hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test
```

➢ **复制本地文件到 HDFS 的目录的 test1.txt**

```
Hadoop fs -copyFromLocal/usr/local/hadoop/README.txt /user/hduser/test/test1.txt
```

➢ **列出 HDFS 目录下的文件**

```
hadoop fs -ls /user/hduser/test
```

运行后，屏幕显示界面如图 6-10 所示。

```
hduser@master: ~
hduser@master:~$ hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test
hduser@master:~$
hduser@master:~$ hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test/test1.txt
hduser@master:~$
hduser@master:~$ hadoop fs -ls /user/hduser/test
Found 2 items
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 11:49 /user/hduser/test/README.txt
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 11:49 /user/hduser/test/test1.txt
hduser@master:~$
```

1. 运行复制命令

2. 查看已经复制的 HDFS 文件

图 6-10　复制本地文件到 HDFS

步骤 02 列出 HDFS 目录下的文件

我们可以在“终端”程序中输入下列命令：

➢ **列出 HDFS 目录下的文件内容**

```
hadoop fs -cat /user/hduser/test/README.txt
```

运行后，屏幕显示界面如图 6-11 所示。

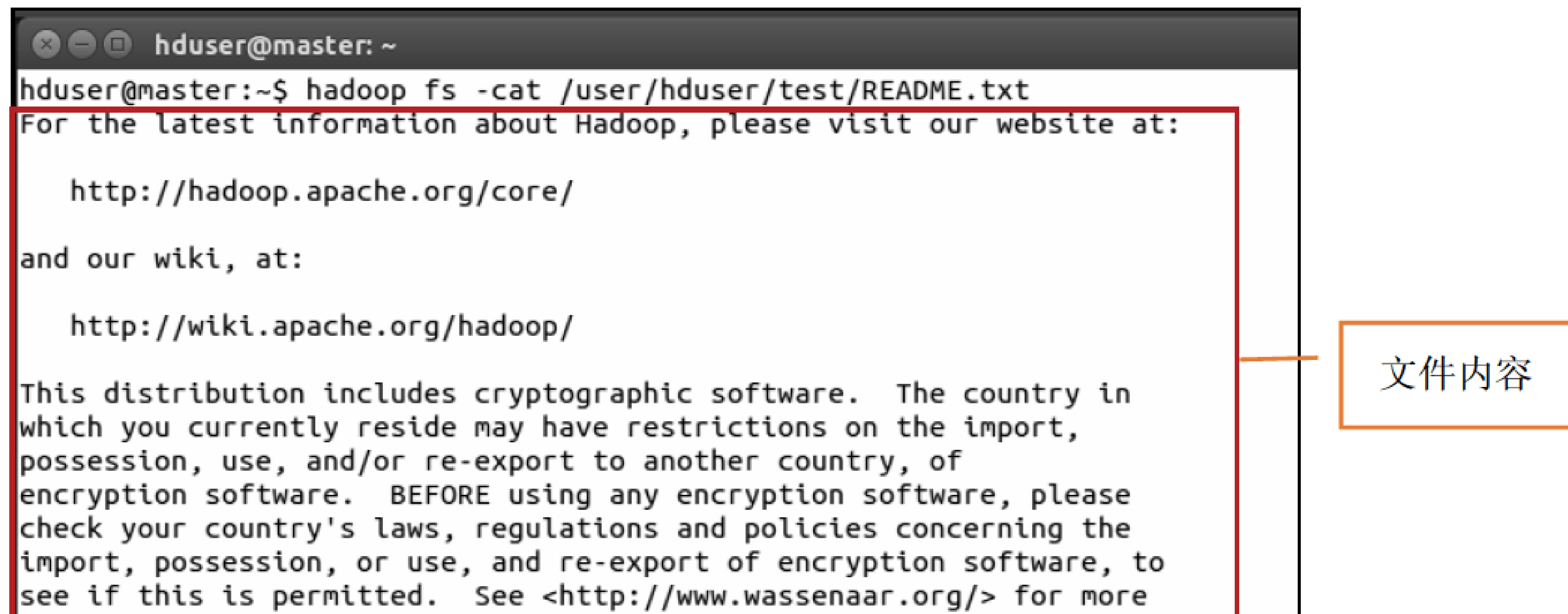


图 6-11　列出 HDFS 目录下的文件

不过，上述命令会一次列出所有文件内容。如果文件太大，可以加上“| more”，一页一页地显示，命令如下：

```
hadoop fs -cat /user/hduser/test/README.txt|more
```

步骤 03 复制重复文件到 HDFS 目录

当我们复制本地文件至 HDFS 目录时，如果文件已经存在，系统会回复“File exists”，即文件已经存在，将不会复制。

➢ **复制本地文件到 HDFS 的目录时，文件已经存在**

```
hadoop fs -copyFromLocal /usr/local/hadoop/README.txt /user/hduser/test
```

运行后屏幕显示界面如图 6-12 所示。

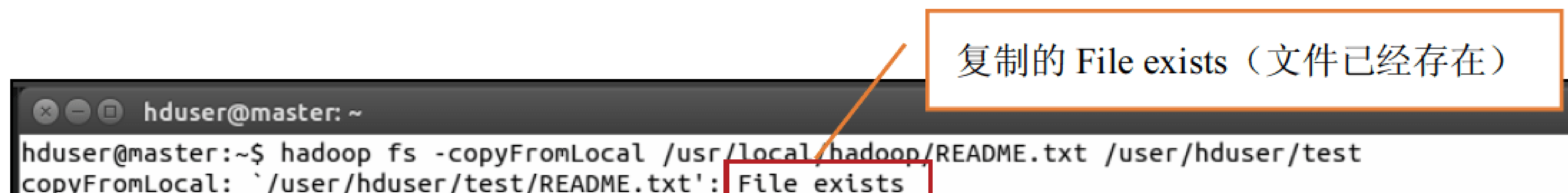


图 6-12　复制重复文件到 HDFS 目录

步骤 04 强制复制重复文件到 HDFS 目录

如果文件已经存在，此时可以加入-f 选项（f 是 force，有强制的意思），命令如下：

> **强制复制文件**

```
hadoop fs -copyFromLocal -f /usr/local/hadoop/README.txt /user/hduser/test
```

强制复制本地文件/usr/local/hadoop/README.txt 到 HDFS 目录/user/hduser/test。

运行后屏幕显示界面如图 6-13 所示。



图 6-13　强制复制文件

步骤 05 复制多个本地文件到 HDFS 目录

也可以一次复制多个本地文件至 HDFS 目录，命令如下：

> **同时复制 NOTICE.txt 与 LICENSE.txt 到 HDFS 目录/user/hduser/test**

```
hadoop fs -copyFromLocal /usr/local/hadoop/NOTICE.txt
 /usr/local/hadoop/LICENSE.txt  /user/hduser/test
```

运行后的屏幕显示界面如图 6-14 所示。

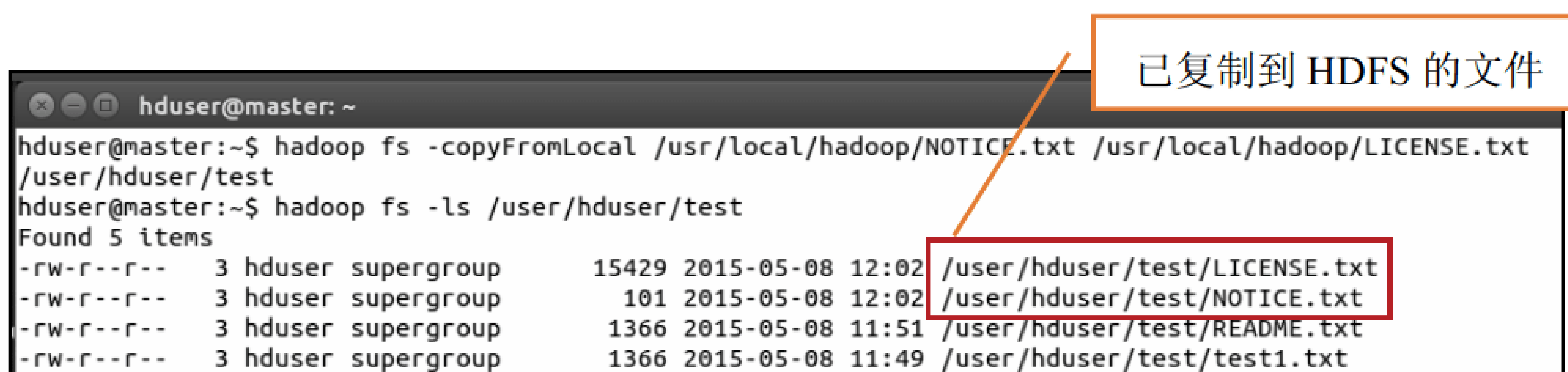


图 6-14　复制多个本地文件到 HDFS 目录

步骤 06 copyFromLocal 复制目录到 HDFS 目录

也可以从本地计算机复制整个目录到 HDFS 目录，命令如下：

> **复制整个本地计算机的目录/usr/local/hadoop/etc 到 HDFS 目录/user/hduser/test**

```
hadoop fs -copyFromLocal /usr/local/hadoop/etc /user/hduser/test
```

> **列出在 HDFS 目录下的文件**

```
hadoop fs -ls /user/hduser/test
```

运行界面如图 6-15 所示。

已经复制到 HDFS 的 etc 目录

```
hduser@master: ~
hduser@master:~$ hadoop fs -copyFromLocal /usr/local/hadoop/etc  /user/hduser/test
hduser@master:~$ hadoop fs -ls /user/hduser/test
Found 5 items
-rw-r--r--   3 hduser supergroup      15429 2015-05-08 12:12 /user/hduser/test/LICENSE.txt
-rw-r--r--   3 hduser supergroup        101 2015-05-08 12:12 /user/hduser/test/NOTICE.txt
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 12:12 /user/hduser/test/README.txt
drwxr-xr-x   - hduser supergroup          0 2015-05-08 12:19 /user/hduser/test/etc
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 12:11 /user/hduser/test/test1.txt
```

图 6-15　复制整个目录到 HDFS 目录

步骤 07　查看目录下所有的文件

之前的命令只是看到 etc 目录名称，还可以使用下列命令：

➢ **列出 HDFS 目录/user/hduser/test/etc 下的所有文件**

```
hadoop fs -ls -R /user/hduser/test/etc
```

以上命令加上-R 选项（-R 是 Recursive）即可列出 HDFS 目录下的所有文件，包含子目录。运行结果的屏幕显示界面如图 6-16 所示。

```
hduser@master:~$ hadoop fs -ls -R /user/hduser/test/etc
drwxr-xr-x   - hduser supergroup          0 2015-05-08 12:19 /user/hduser/test/etc/hadoop
-rw-r--r--   3 hduser supergroup       4436 2015-05-08 12:19 /user/hduser/test/etc/hadoop/capacity-scheduler.xml
-rw-r--r--   3 hduser supergroup       1335 2015-05-08 12:19 /user/hduser/test/etc/hadoop/configuration.xsl
-rw-r--r--   3 hduser supergroup        318 2015-05-08 12:19 /user/hduser/test/etc/hadoop/container-executor.cfg
-rw-r--r--   3 hduser supergroup       1046 2015-05-08 12:19 /user/hduser/test/etc/hadoop/core-site.xml
-rw-r--r--   3 hduser supergroup        872 2015-05-08 12:19 /user/hduser/test/etc/hadoop/core-site.xml~
-rw-r--r--   3 hduser supergroup       3670 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-env.cmd
-rw-r--r--   3 hduser supergroup       4246 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-env.sh
-rw-r--r--   3 hduser supergroup       4224 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-env.sh~
-rw-r--r--   3 hduser supergroup       2490 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-metrics.properties
-rw-r--r--   3 hduser supergroup       2598 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-metrics2.properties
-rw-r--r--   3 hduser supergroup       9683 2015-05-08 12:19 /user/hduser/test/etc/hadoop/hadoop-policy.xml
```

已经复制到 HDFS 的 etc 目录下的所有文件

图 6-16　查看目录下的所有文件

步骤 08　使用 put 复制文件到 HDFS 目录

另外，也可以使用“–put”选项复制文件到 HDFS 目录。使用“-put”与“-copyFromLocal”的不同之处是：如果文件已经存在，系统不会显示文件已经存在，而会直接覆盖，例如下列命令：

➢ **使用 put 复制文件到 HDFS 目录，会直接覆盖文件**

```
hadoop fs -put /usr/local/hadoop/README.txt /user/hduser/test/test2.txt
```

运行后屏幕显示界面如图 6-17 所示。

```
hduser@master:~$ hadoop fs -put /usr/local/hadoop/README.txt /user/hduser/test/test2.txt
hduser@master:~$ hadoop fs -ls /user/hduser/test
Found 6 items
-rw-r--r--   3 hduser supergroup      15429 2015-05-08 12:12 /user/hduser/test/LICENSE.txt
-rw-r--r--   3 hduser supergroup        101 2015-05-08 12:12 /user/hduser/test/NOTICE.txt
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 12:12 /user/hduser/test/README.txt
drwxr-xr-x   - hduser supergroup          0 2015-05-08 12:19 /user/hduser/test/etc
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 12:11 /user/hduser/test/test1.txt
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 12:28 /user/hduser/test/test2.txt
hduser@master:~$
```

已经复制至 HDFS 的 test2.txt 文件

图 6-17　使用 put 复制文件

步骤 09　使用 put 命令接受 stdin（标准输入）

另外一个使用-put 与-copyFromLocal 的不同处是："-put"可以接受 stdin（标准输入），例如下列命令：

➢ **将原本显示在屏幕上的内容存储到 HDFS 文件**

```
echo abc | hadoop fs -put -  /user/hduser/test/echoin.txt
```

echo abc 原本是要显示在屏幕上的内容，现在通过"|"（pipe 管道）符号传递给 hadoop 的命令，并且存储到 HDFS 目录下的文件 echoin.txt 中。

➢ **显示在 HDFS 中 echoin.txt 文件的内容**

```
hadoop fs -cat /user/hduser/test/echoin.txt
```

运行后列出 abc，如图 6-18 所示。

```
hduser@master:~$ echo abc | hadoop fs -put -  /user/hduser/test/echoin.txt
hduser@master:~$ hadoop fs -cat /user/hduser/test/echoin.txt
abc
```

echoin.txt 的文件内容

图 6-18　显示文件内容

步骤 10　使用 put 命令将本地目录的列表存储到 HDFS 文件

➢ **将本地目录的列表存储到 HDFS 文件**

```
ls /usr/local/hadoop | hadoop fs -put - /user/hduser/test/hadooplist.txt
```

原本 ls /usr/local/hadoop 命令会把本地目录的列表列在屏幕上，通过"|"符号（pipe 管道）传递给了 Hadoop 命令，所以最后会存储到 HDFS 目录下的 hadooplist.txt 文件中。

➢ **显示 HDFS 中 hadooplist.txt 文件的内容**

```
hadoop fs -cat /user/hduser/test/hadooplist.txt
```

运行后屏幕显示界面如图 6-19 所示。



图 6-19　将本地目录的列表存储到 HDFS

6.4 将 HDFS 上的文件复制到本地计算机

将 HDFS 上的文件复制到本地计算机的示意图如图 6-20 所示。

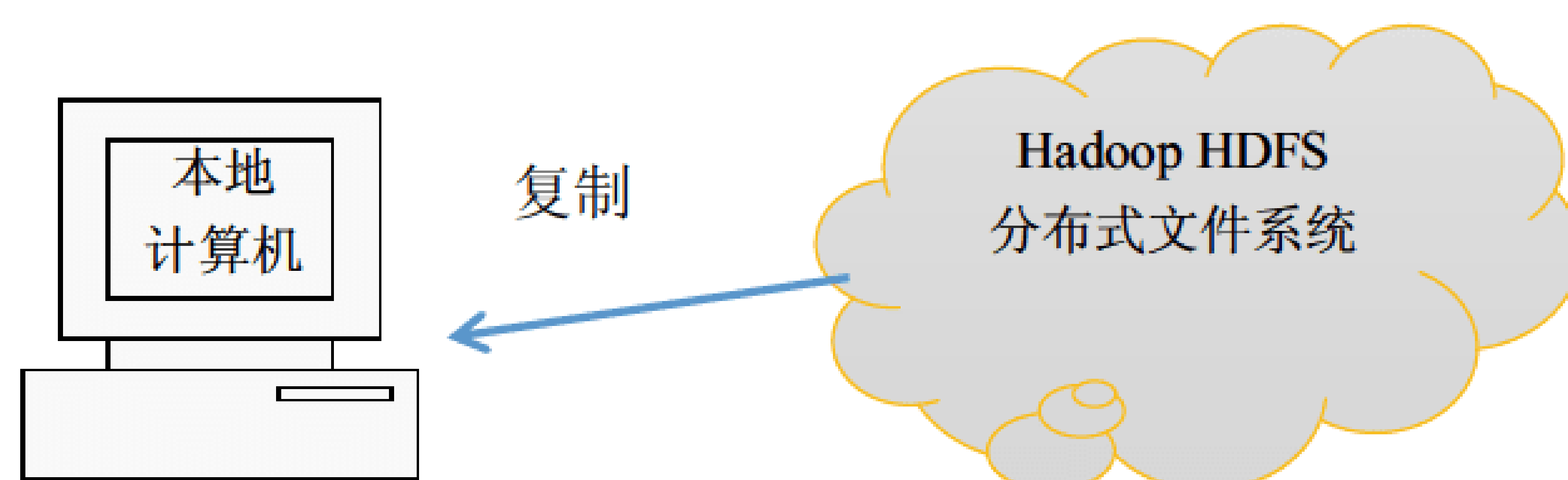


图 6-20　示意图

步骤 01　将 HDFS 上的文件复制到本地计算机（local）

在“终端”程序中输入下列命令，将 HDFS 上的文件复制到本地。

➢ **在本地计算机创建 test 测试目录**

```
mkdir test
```

➢ **切换到 test 目录**

```
cd test
```

➢ **将 HDFS 的文件复制到本地计算机**

```
hadoop fs -copyToLocal /user/hduser/test/hadooplist.txt
```

➢ **查看本地目录**

```
ll
```

运行后的屏幕显示界面如图 6-21 所示。

```
hduser@master: ~/test
hduser@master:~$ mkdir test
hduser@master:~$ cd test
hduser@master:~/test$ hadoop fs -copyToLocal  /user/hduser/test/hadooplist.txt
hduser@master:~/test$ ll
总用量 12
drwxrwxr-x  2 hduser hduser 4096  5月  8 20:03 ./
drwxr-xr-x 25 hduser hduser 4096  5月  8 20:03 ../
-rw-r--r--  1 hduser hduser   90  5月  8 20:03 hadooplist.txt
```

已经复制到本地计算机的 hadooplist.txt 文件

图 6-21　将 HDFS 上的文件复制到本地计算机

步骤 02 将 HDFS 上的目录复制到本地计算机

也可以使用下列命令：

➢ **将整个 HDFS 上的目录复制到本地计算机**

```
hadoop fs -copyToLocal  /user/hduser/test/etc
```

运行后屏幕显示界面如图 6-22 所示。

```
hduser@master: ~/test
hduser@master:~/test$ hadoop fs -copyToLocal  /user/hduser/test/etc
hduser@master:~/test$ ll
总用量 16
drwxrwxr-x  3 hduser hduser 4096  5月  8 20:07 ./
drwxr-xr-x 25 hduser hduser 4096  5月  8 20:03 ../
drwxrwxr-x  3 hduser hduser 4096  5月  8 20:07 etc/
-rw-r--r--  1 hduser hduser   90  5月  8 20:03 hadooplist.txt
```

已经复制到本地的目录 etc

图 6-22　将 HDFS 上的目录复制到本地计算机

步骤 03 hadoop fs -get 复制到本地计算机（local）

也可以使用下列“-get”命令：

➢ **将 HDFS 上的文件复制到本地计算机（local）**

```
hadoop fs -get  /user/hduser/test/README.txt localREADME.txt
```

运行后屏幕显示界面如图 6-23 所示。

```
hduser@master: ~/test
hduser@master:~/test$ hadoop fs -get  /user/hduser/test/README.txt localREADME.txt
hduser@master:~/test$ ll
总用量 20
drwxrwxr-x  3 hduser hduser 4096  5月  8 20:12 ./
drwxr-xr-x 25 hduser hduser 4096  5月  8 20:03 ../
drwxrwxr-x  3 hduser hduser 4096  5月  8 20:07 etc/
-rw-r--r--  1 hduser hduser   90  5月  8 20:03 hadooplist.txt
-rw-r--r--  1 hduser hduser 1366  5月  8 20:12 localREADME.txt
```

已经复制到本地的目录 localREADME.txt

图 6-23　使用“-get”命令复制到本地计算机

6.5　复制与删除 HDFS 文件

如图 6-24 所示复制 HDFS 文件，指的是在 HDFS 中复制文件或目录到另一个 HDFS 目录。

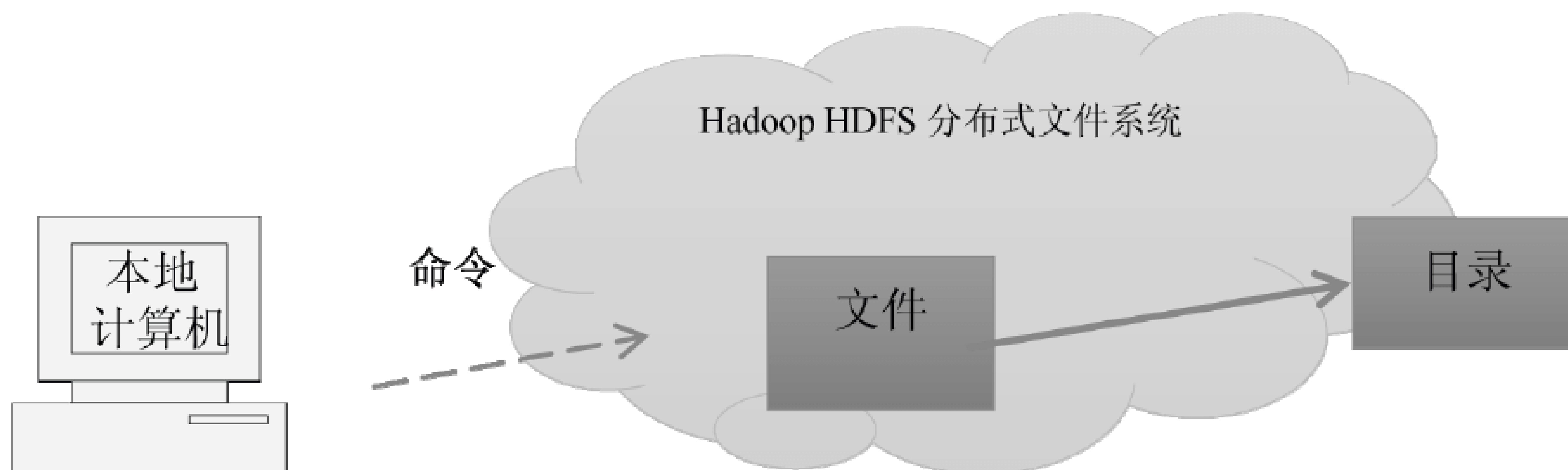


图 6-24　复制 HDFS 文件的示意图

步骤 01　复制 HDFS 文件

在“终端”程序中输入下列命令，在 HDFS 中复制文件或目录到另一个 HDFS 目录：

➢ **在 HDFS 上创建测试目录/user/hadoop/test/temp**

```
hadoop fs -mkdir /user/hduser/test/temp
```

➢ **复制 HDFS 文件到 HDFS 测试目录**

```
hadoop fs -cp /user/hduser/test/README.txt /user/hduser /test/temp
```

复制 HDFS 文件/user/hduser/test/README.txt 到 HDFS 测试目录/user/hadoop/test/temp。

➢ **查看 HDFS 测试目录**

```
hadoop fs -ls /user/hduser/test/temp
```

运行后屏幕显示界面如图 6-25 所示。

```
hduser@master: ~/test
hduser@master:~/test$ hadoop fs -mkdir /user/hduser/test/temp
hduser@master:~/test$ hadoop fs -cp /user/hduser/test/README.txt /user/hduser/test/temp
hduser@master:~/test$ hadoop fs -ls  /user/hduser/test/temp
Found 1 items
-rw-r--r--   3 hduser supergroup       1366 2015-05-08 20:59 /user/hduser/test/temp/README.txt
hduser@master:~/test$ 
```

图 6-25　复制 HDFS 文件

步骤 02　删除 HDFS 文件

在“终端”程序中输入下列命令，删除 HDFS 文件：

➢ **先查看准备要被删除的文件**

```
hadoop fs -ls /user/hduser/test
```

➢ **删除 HDFS 文件**

```
hadoop fs -rm /user/hduser/test/test2.txt
```

运行后屏幕显示界面如图 6-26 所示。

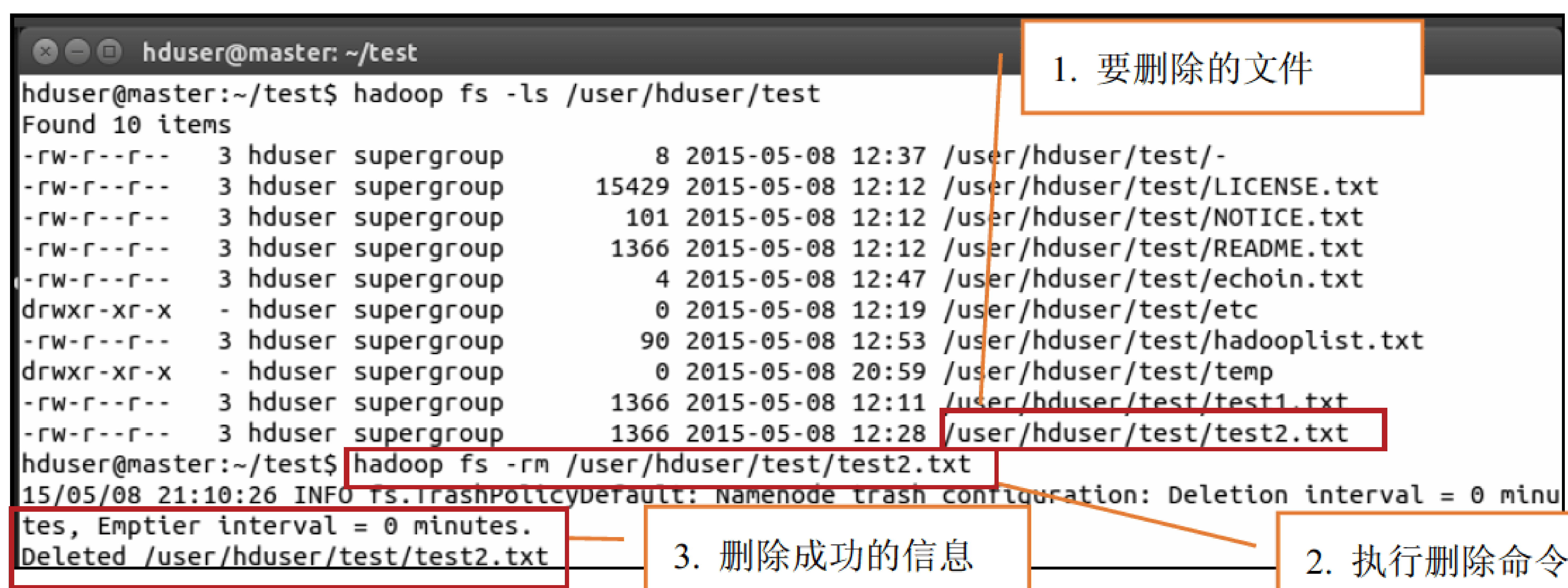


图 6-26　删除 HDFS 文件

步骤 03 删除 HDFS 目录

在“终端”程序中输入下列命令，删除 HDFS 目录：

➢ **查看准备要被删除的 HDFS 目录**

```
hadoop fs -ls  /user/hduser/test
```

➢ **删除 HDFS 目录**

```
hadoop fs -rm -R /user/hduser/test/etc
```

运行后屏幕显示界面如图 6-27 所示。

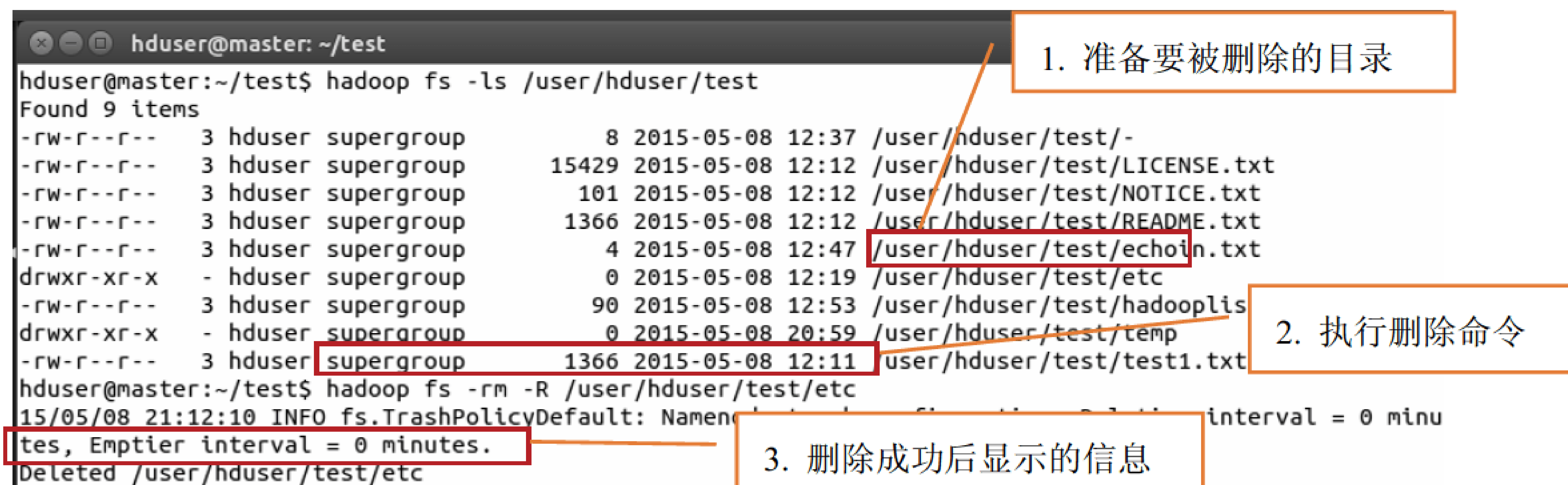


图 6-27　删除 HDFS 目录

6.6 在 Hadoop HDFS Web 用户界面浏览 HDFS

之前我们使用 HDFS 命令来查看 HDFS 目录。除此之外，Hadoop 还提供了更方便的功能，可以在 Hadoop HDFS Web 用户界面来浏览 HDFS 目录或文件，如图 6-28 所示。

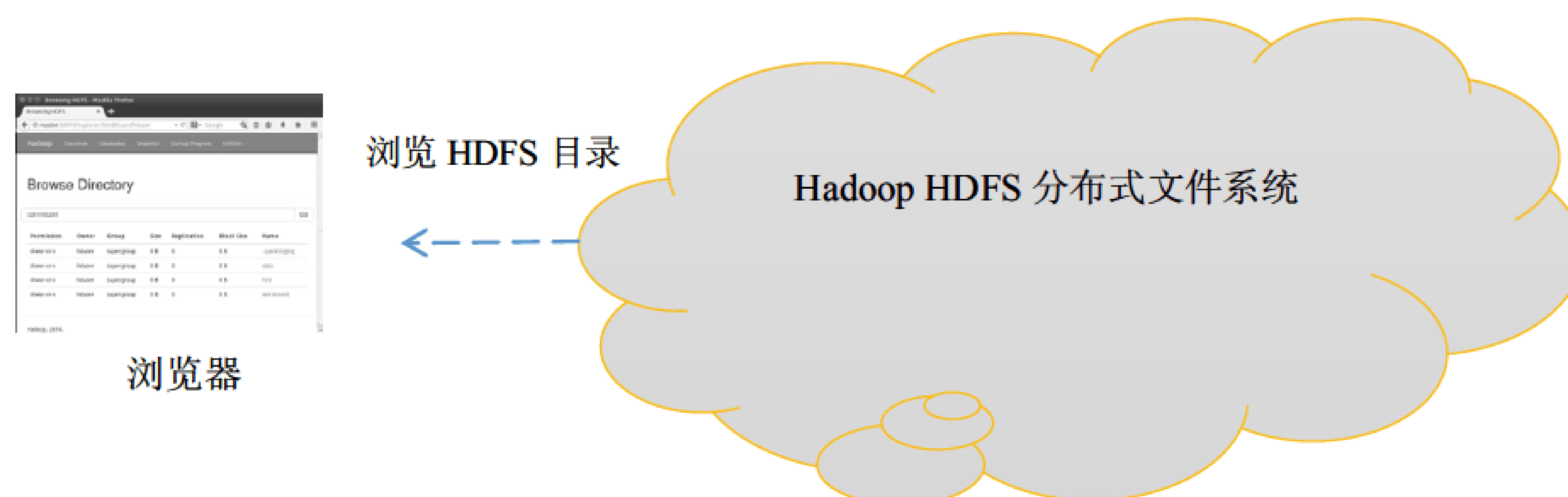


图 6-28　在 Hadoop HDFS Web 用户界面浏览 HDFS

步骤 01 浏览 Hadoop HDFS Web 界面查看 HDFS

网址	说明
http://master:50070	Hadoop HDFS Web 界面网址

参照如图 6-29 所示的步骤操作就可以看到我们之前创建的 HDFS 目录与文件。



图 6-29　浏览 Hadoop HDFS Web 界面查看 HDFS

步骤 02 输入完整的目录路径来查看文件

也可以直接输入完整的目录路径来查看文件，如图 6-30 所示。



图 6-30 输入完整的目录路径来查看文件

步骤 03 查看 HDFS 文件的完整信息

单击文件名链接后，会打开 File Information 对话框显示此文件的信息，如图 6-31 所示。

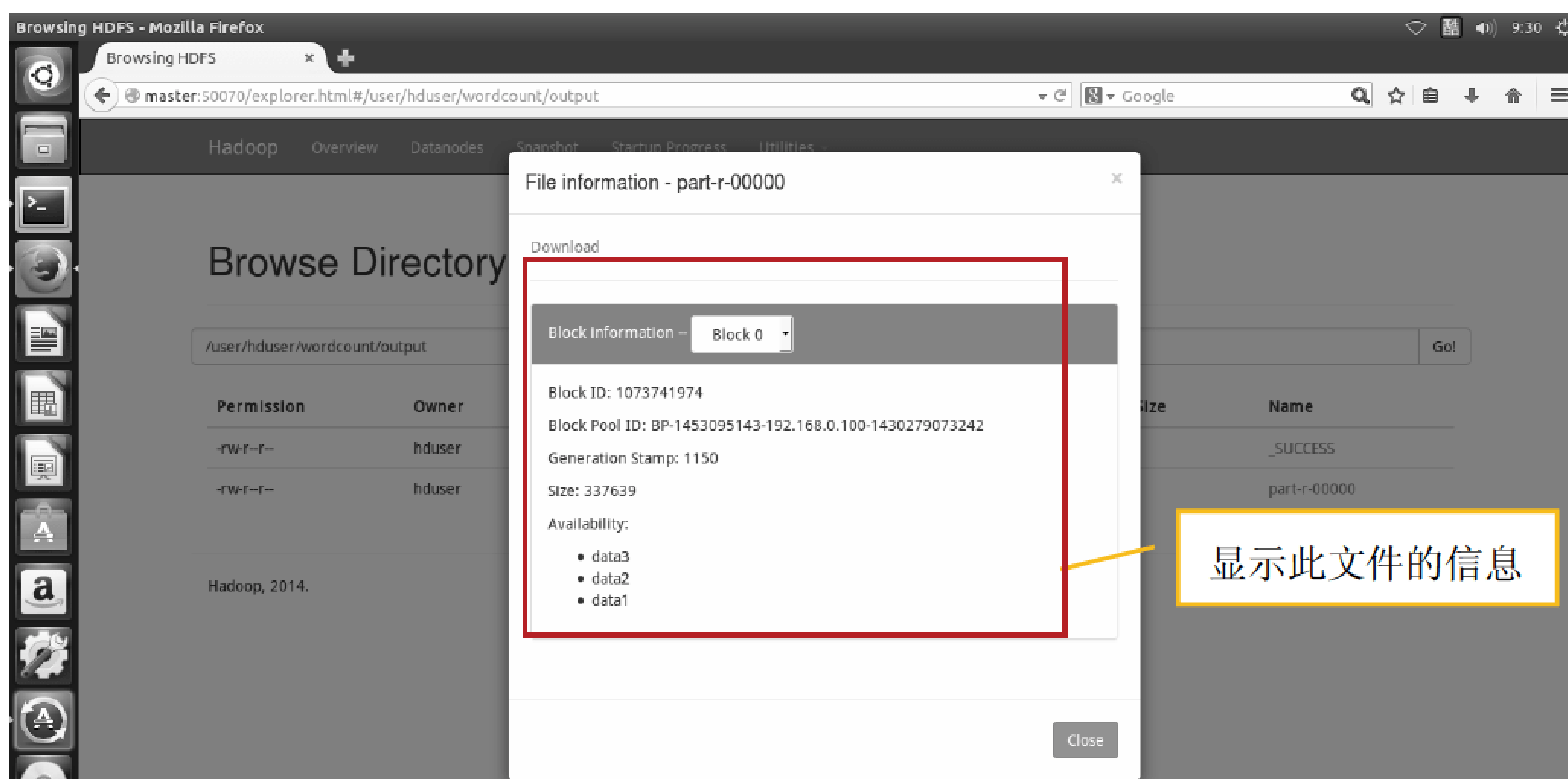


图 6-31 查看 HDFS 文件的完整信息

步骤 04 下载 HDFS 文件

可以参照图 6-32 所示的步骤来下载 HDFS 文件。



图 6-32　下载 HDFS 文件

步骤 05 打开下载文件

下载文件后可以查看下载的文件，如图 6-33 所示。

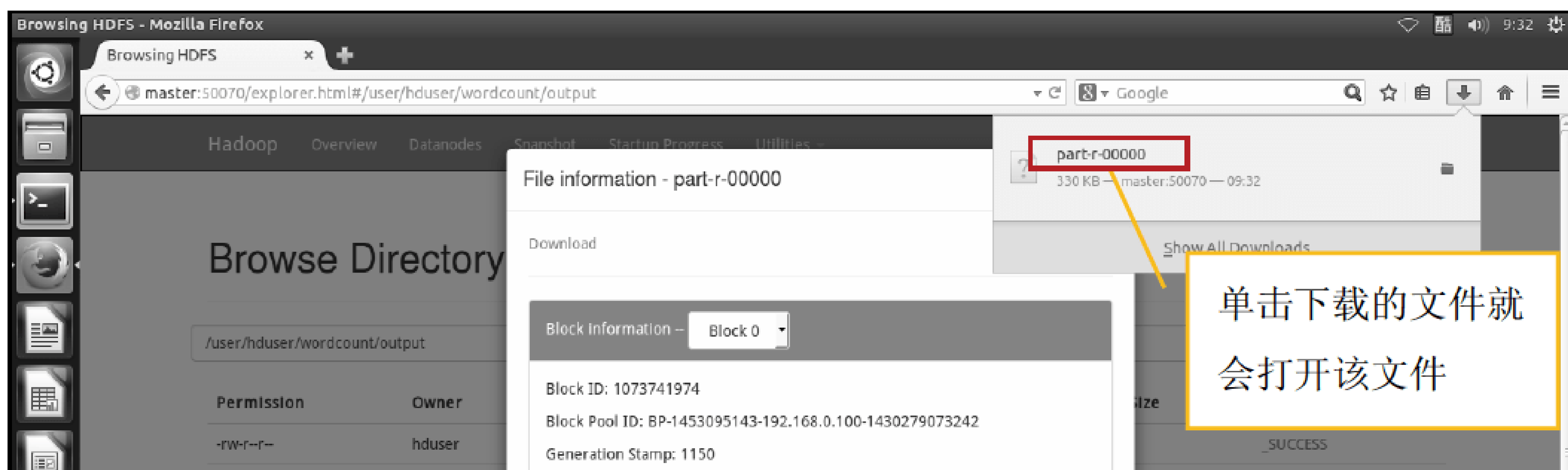


图 6-33　打开下载文件

步骤 06 打开文件

因为是文本文件，所以默认会使用 gedit 打开文件，屏幕显示界面如图 6-34 所示。

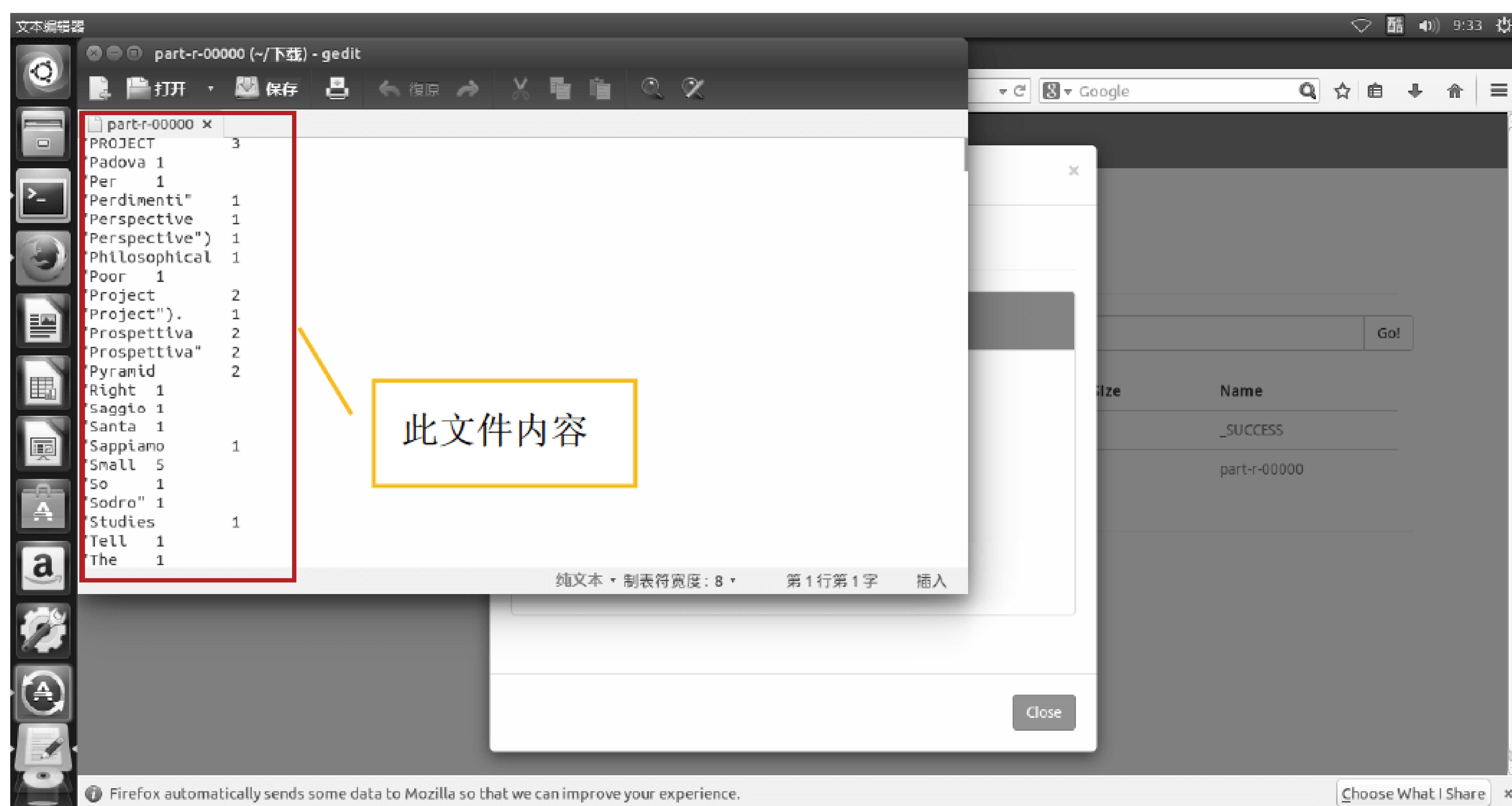


图 6-34　打开文件

6.7 结论

本章介绍了 Hadoop HDFS 分布式存储，并且在实际计算机上操作了 HDFS 常用命令，这些命令在后续章节都会使用。在下一章中，我们将介绍 Hadoop Map/ Reduce 分布式计算。

第 7 章

Hadoop MapReduce

MapReduce 是一种程序开发模式，可以使用大量服务器来并行处理。MapReduce，简单地说，Map 就是分配工作，Reduce 就是将工作结果汇总整理。

- 首先使用 Map 将待处理的数据分割成很多的小份数据，由每台服务器分别运行。
- 再通过 Reduce 程序进行数据合并，最后汇总整理出结果。

本章将以 WordCount.Java 作为范例来介绍 MapReduce。

7.1 简单介绍 WordCount.java

下列为 WordCount 范例，要计算文件中每一个英文单词出现的次数，步骤如图 7-1 所示。

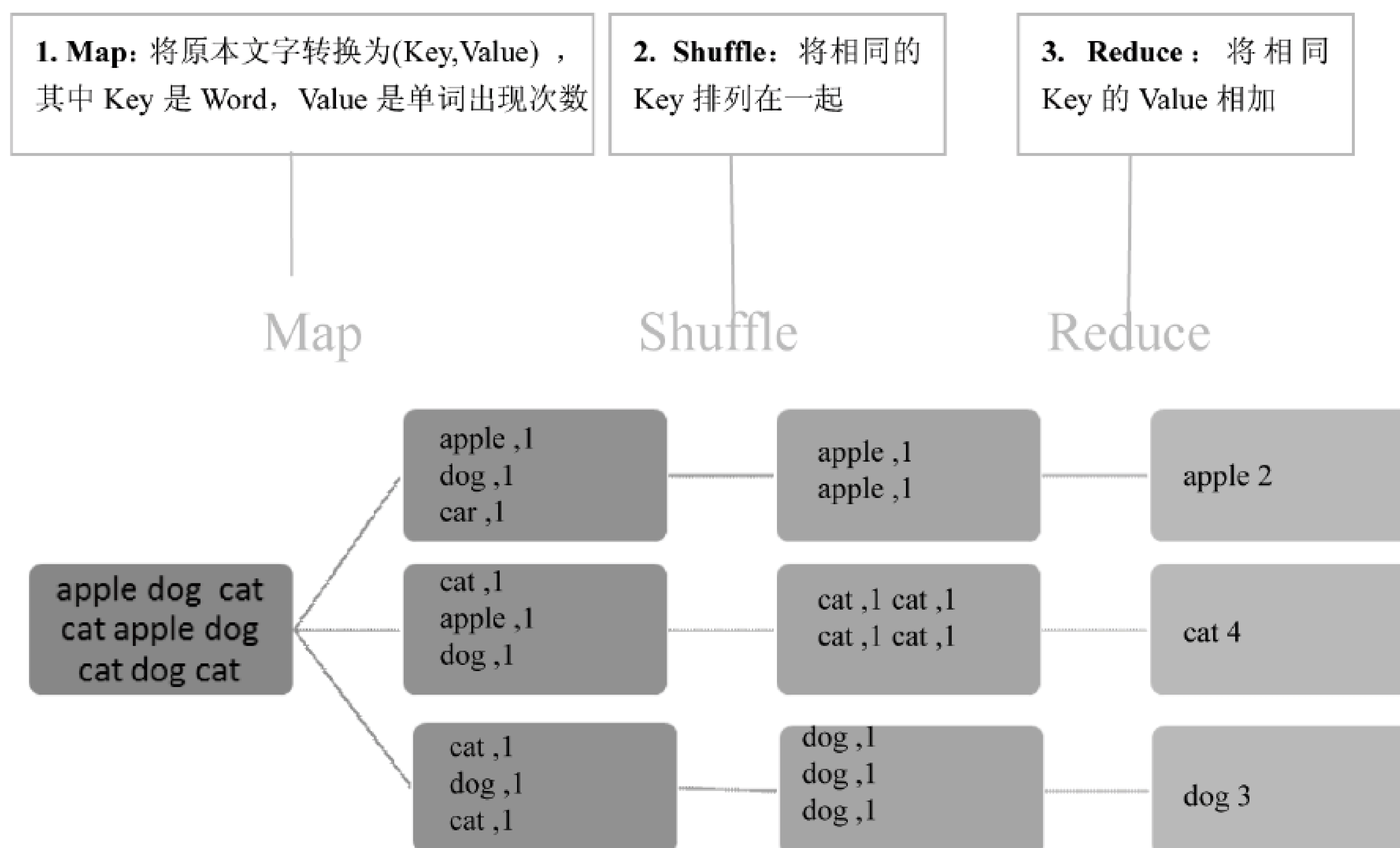


图 7-1 范例程序计算英文单词出现次数的大致步骤

本章将以 WordCount.java 示范 Map Reduce 的运行方式。

➢ WordCount 开发步骤如表 7-1 所示

本章所有命令都在 master 虚拟机的“终端”程序中运行。

表 7-1 WordCount 开发步骤

顺序	开发步骤	说明
1	编辑 WordCount.java	使用 gedit 编辑 WordCount.java
2	编译 WordCount.java	编译、打包 WordCount.java 程序
3	创建测试文本文件	使用 Hadoop 目录下的 LICENSE.txt 文件作为测试文件，并上传文本文件至 HDFS
4	运行 WordCount.java	在 Hadoop 环境运行 WordCount
5	查看运行结果	运行会产生输出文件并存储到 HDFS 中，可使用 HDFS 命令查看

➢ 本章命令整理

本章命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可节省打字的时间，也不用担心打错字（无法在 VirtualBox

虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

7.2　编辑 WordCount.java

在此我们使用最简单的方式（使用 gedit）编辑 WordCount.java。

步骤 01　创建 wordcount 目录

在“终端”程序中输入下列命令，创建 wordcount 测试目录：

➢ 创建 wordcount 测试目录

```
mkdir -p ~/wordcount/input
```

➢ 切换至 wordcount 测试目录

```
cd  ~/wordcount
```

运行后屏幕显示界面如图 7-2 所示。

```
hduser@master: ~/wordcount
hduser@master:~$ mkdir -p ~/wordcount/input
hduser@master:~$ cd  ~/wordcount
hduser@master:~/wordcount$
```

图 7-2　创建 wordcount 目录

步骤 02　复制 WordCount.java 程序代码

在 Hadoop 说明文件中就具有 WordCount.java 的程序代码，在浏览器输入下列网址就可以看到原文的详细说明与源代码。后续操作我们可以复制这个网页中的源代码（见图 7-3）：

```
http://hadoop.apache.org/docs/current/hadoop-MapReduce-client/hadoop-MapReduce-client-core/MapReduceTutorial.html
```

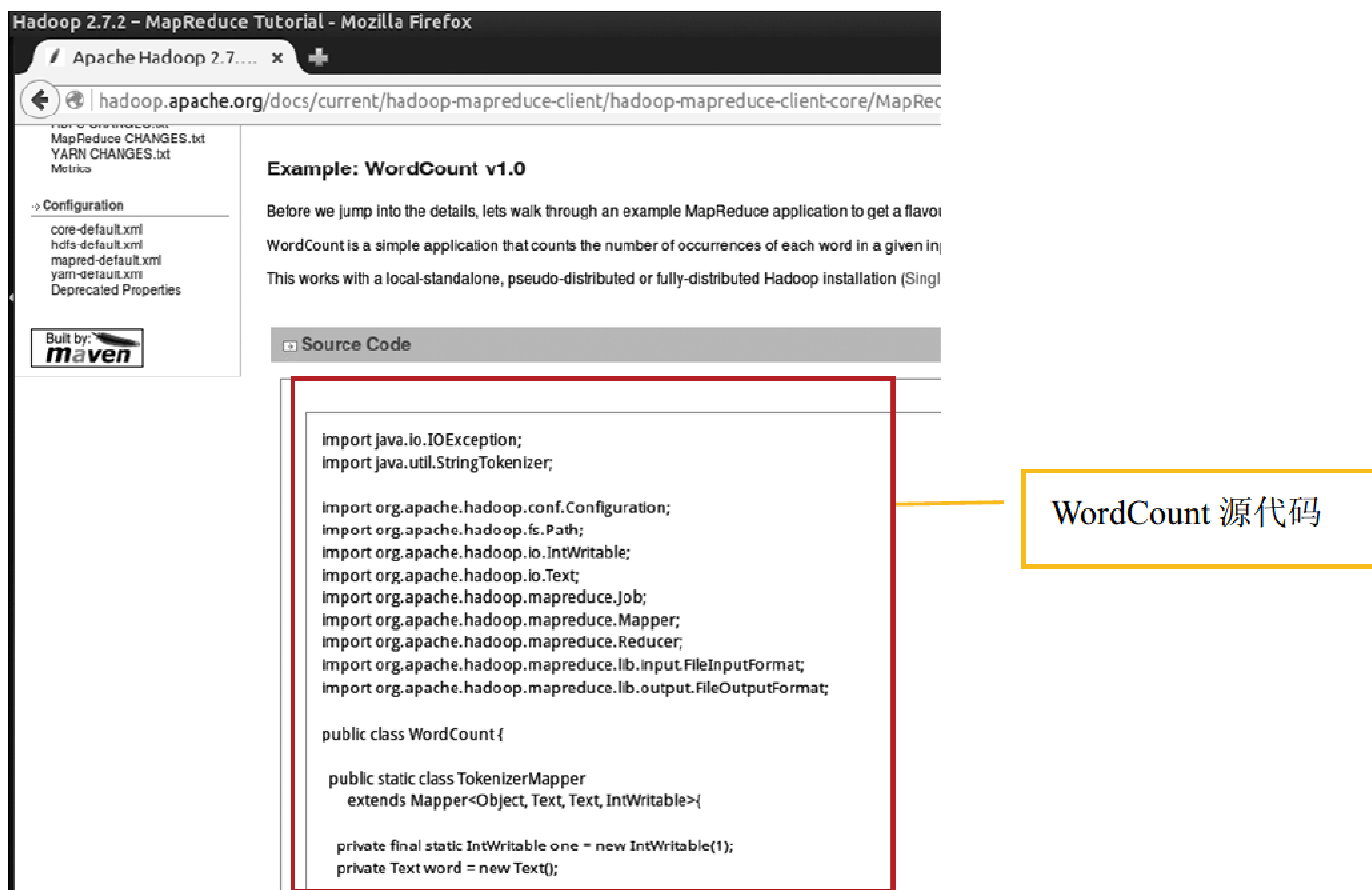


图 7-3　复制 WordCount.java 程序代码

步骤 03　编辑 WordCount.java

在“终端”程序中输入下列命令：

➢ **使用 gedit 编辑 WordCount.java**

```
sudo gedit WordCount.Java
```

按 Enter 键之后就会打开 WordCount.java 空白文件，如图 7-4 所示。



图 7-4　编辑 WordCount.java

步骤 04　Import 相关的 Lib 并创建 WordCount 类

首先要 Import 相关的 Lib 链接库，并且创建 WordCount 类 class，后续的程序都是引入到 WordCount 类中，如图 7-5 所示。

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

}
```

1. Import 相关的 Lib 链接库

2. WordCount 类

图 7-5 Import 相关的 Lib 并创建 WordCount 类

步骤 05 创建 main function

接下来创建 main function 程序。main function 是程序的起点，在 main function 中，主要设置 map 与 reduce 类，如图 7-6 所示。

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

1. 设置 TokenizerMapper 类

2. 设置 IntSumReducer 类

图 7-6 创建 main function

步骤 06 创建 TokenizerMapper 类

在 TokenizerMapper 类的 map 方法中，先创建 StringTokenizer，然后使用 while (itr.hasMoreTokens()) 读取每一个 word 并写入 context.write(word, one)，创建 key/value (word, 1)，如图 7-7 所示。

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
```

图 7-7 创建 TokenizerMapper 类

步骤07 创建 IntSumReducer 类

在 IntSumReducer 类的 reduce 方法中，先使用 for (IntWritable val : values)读取每一个数值，并使用 sum += val.get() 求和，最后写入结果，如图 7-8 所示。

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

图 7-8　创建 IntSumReducer 类

步骤08 编辑完成的 wordCount.java

编辑完成后，先单击“保存”按钮再单击“关闭”按钮，如图 7-9 所示。

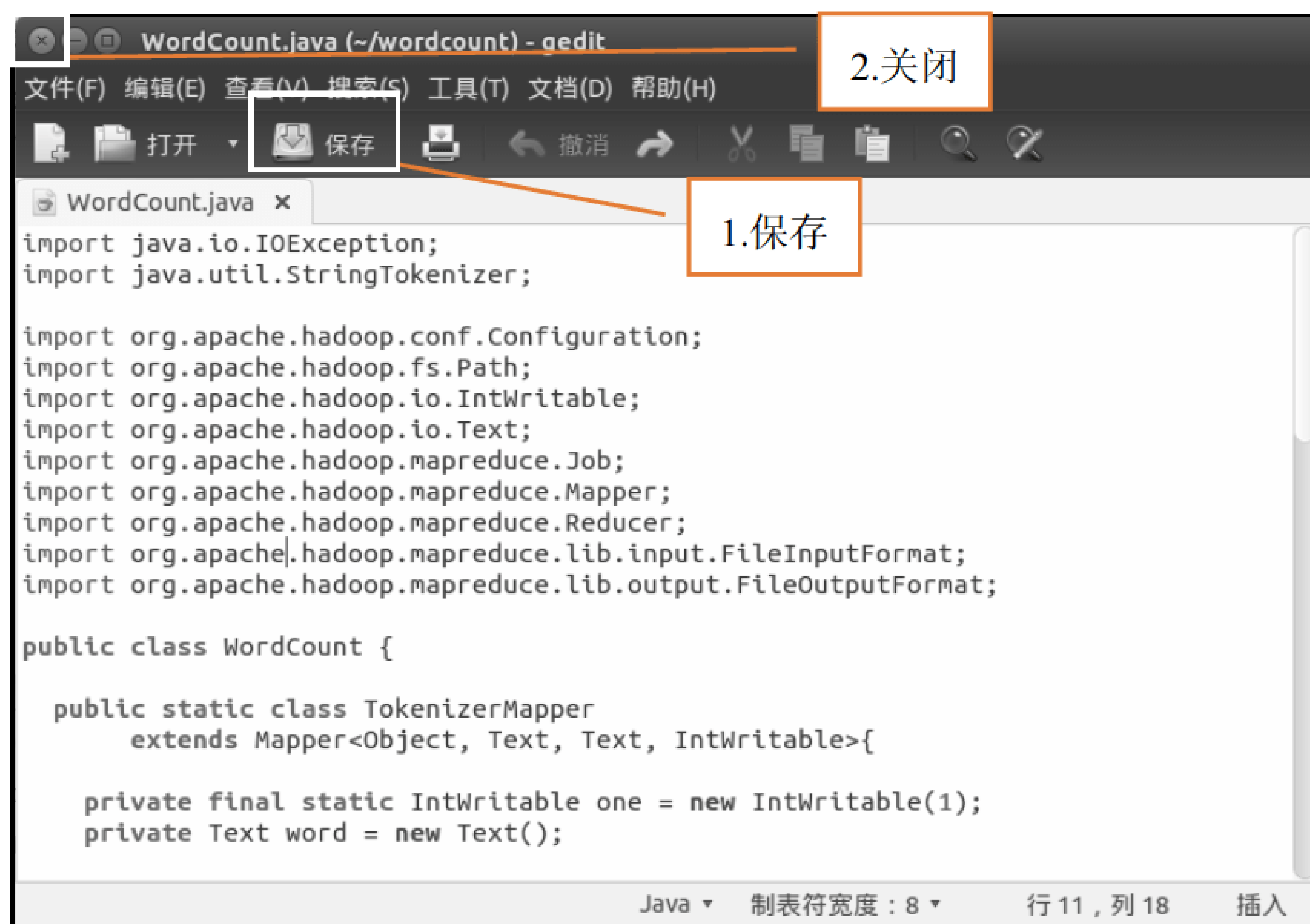


图 7-9　编辑完成后保存

步骤09 查看编辑完成的 WordCount.java

保存后，可以使用以下命令：

➢ **查看之前编辑完成的 WordCount.java**

```
ll
```

运行后，屏幕显示界面如图 7-10 所示。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ ll
总用量 16
drwxrwxr-x  3 hduser hduser 4096  5月 15 20:19 ./
drwxr-xr-x 21 hduser hduser 4096  5月 15 19:34 ../
drwxrwxr-x  2 hduser hduser 4096  5月 15 19:48 input/
-rw-r--r--  1 root   root   2091  5月 15 19:33 WordCount.java
hduser@master:~/wordcount$
```

编辑完成的 WordCount.java

图 7-10　查看 WordCount.java

7.3 编译 WordCount.java

接下来编译 WordCount.java，只是需要先设置环境变量。

步骤 01 修改编译所需要的环境变量文件

在“终端”程序中输入下列命令：

➢ **编辑~/.bashrc**

```
sudo gedit ~/.bashrc
```

输入后按 Enter 键就会打开~/.bashrc，如图 7-11 所示。



图 7-11　修改环境变量文件

上述命令说明如下：

➤ **设置 PATH**

```
export PATH=${JAVA_HOME}/bin:${PATH}
```

因为我们后续要使用的命令（例如 jar）存在于/usr/lib/jvm/java-7-openjdk-amd64/bin，所以将${JAVA_HOME}/bin 加入 PATH，这样，切换到其他目录时也就能够使用此命令了。

➤ **设置 HADOOP_CLASSPATH，编译时才找得到链接库**

```
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

步骤 02 让 ~/.bashrc 修改的设置值生效

当我们修改~/.bashrc 之后，可以先从系统注销再重新登录，该设置就会生效；或使用 source 命令让~/.bashrc 设置立即生效。

在“终端”程序中输入下列命令：

➤ **让~/.bashrc 设置生效**

```
source ~/.bashrc
```

屏幕显示界面如图 7-12 所示。



图 7-12 让~/.bashrc 修改的设置值生效

步骤 03 开始编译

你可以使用下列命令进行编译，编译完成后，就会生成 wc.jar 文件。

请在“终端”程序中输入表 7-2 所示的命令。

表 7-2 编译命令

命令	说明
hadoop com.sun.tools.javac.Main WordCount.java	编译 WordCount 程序
jar cf wc.jar WordCount*.class	把 WordCount 类打包成 wc.jar
ll	查看目录内容

运行结果如图 7-13 所示。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop com.sun.tools.javac.Main WordCount.java
hduser@master:~/wordcount$ jar cf wc.jar WordCount*.class
hduser@master:~/wordcount$ ll
总用量 32
drwxrwxr-x  3 hduser hduser 4096  5月 15 19:36 ./
drwxr-xr-x 21 hduser hduser 4096  5月 15 19:34 ../
drwxrwxr-x  2 hduser hduser 4096  5月 15 19:26 input/
-rw-rw-r--  1 hduser hduser 3072  5月 15 19:36 wc.jar
-rw-rw-r--  1 hduser hduser 1501  5月 15 19:36 WordCount.class
-rw-rw-r--  1 hduser hduser 1739  5月 15 19:36 WordCount$IntSumReducer.class
-rw-r--r--  1 root   root   2091  5月 15 19:33 WordCount.java
-rw-rw-r--  1 hduser hduser 1736  5月 15 19:36 WordCount$TokenizerMapper.class
hduser@master:~/wordcount$
```

生成的 wc.jar 文件

图 7-13　编译结果

jar 文件（java ARchive）是一种压缩文件，包含 Class 与相关资源（文字、图片等），之后就可以运行 wc.jar 这个程序了。

7.4 创建测试文本文件

为了测试 WordCount 程序，我们可以使用 Hadoop 目录下的 LICENSE.txt 文件。

步骤 01 复制 LICENSE.txt

在“终端”程序中输入下列命令，下载文本文件：

➢ 切换到输入目录

```
cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
ll ~/wordcount/input
```

运行结果如图 7-14 所示。

```
hduser@master: ~
hduser@master:~$ cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
hduser@master:~$ ll ~/wordcount/input
总用量 544
drwxrwxr-x 2 hduser hduser   4096  5月 15 19:48 ./
drwxrwxr-x 3 hduser hduser   4096  5月 15 19:36 ../
-rw-r--r-- 1 hduser hduser  15429  5月 15 19:48 LICENSE.txt
```

图 7-14　复制 LICENSE.txt

步骤 02 启动所有虚拟服务器

首先，必须要启动所有虚拟服务器 master、data1、data2、data3，如图 7-15 所示。



图 7-15 启动所有虚拟服务器

步骤 03 进入 master 虚拟机，启动 Hadoop Multi-Node Cluster

在 master 虚拟机启动“终端”程序，输入下列指令：

➢ 启动 Hadoop Multi-Node Cluster

```
Start-all.sh
```

运行后如图 7-16 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namen
ode-master.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data2.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-master.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noder
anager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noder
anager-data3.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noder
anager-data1.out
hduser@master:~$
```

图 7-16 启动 Hadoop Multi-Node Cluster

步骤 04 上传测试文件到 HDFS 目录

接下来，我们将之前下载的文本文件从本地复制到 HDFS。

在“终端”程序中输入下列命令：

➢ 在 HDFS 创建目录

```
hadoop fs -mkdir -p /user/hduser/wordcount/input
```

➢ **切换到~/wordcount/input 数据文件目录**

```
cd ~/wordcount/input
```

➢ **上传文本文件到 HDFS**

```
hadoop fs -copyFromLocal LICENSE.txt /user/hduser/wordcount/input
```

➢ **列出 HDFS 文件**

```
hadoop fs -ls /user/hduser/wordcount/input
```

运行的结果如图 7-17 所示，已将 LICENSE.txt 上传到 HDFS 的/user/hduser/wordcount/input 目录。

```
hduser@master: ~/wordcount/input
hduser@master:~$ hadoop fs -mkdir -p /user/hduser/wordcount/input
hduser@master:~$ cd ~/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -copyFromLocal LICENSE.txt /user/hdus
er/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -ls /user/hduser/wordcount/input
Found 2 items
-rw-r--r--   3 hduser supergroup      15429 2016-05-15 19:57 /user/hduser/wordco
unt/input/LICENSE.txt
```

图 7-17　上传测试文件到 HDFS 目录

7.5　运行 WordCount.java

在“终端”程序中输入下列命令，运行 WordCount 程序：

➢ **切换目录**

```
cd ~/wordcount
```

➢ **运行 WordCount 程序**

```
hadoop jar wc.jar WordCount /user/hduser/wordcount/input/LICENSE.txt /user/
hduser/wordcount/output
```

运行 WordCount 程序，语句格式为“hadoop jar wc.jar [输入文件] [输出目录]”。如上述命令所示，程序会读取输入文件/user/hduser/wordcount/input/LICENSE.txt，处理完成后将结果写入/user/hduser/wordcount/output 目录。

运行的结果如图 7-18 所示。

```
hduser@master: ~/wordcount
hduser@master:~$ cd ~/wordcount
hduser@master:~/wordcount$ hadoop jar wc.jar WordCount /user/hduser/wordcount/input/LICENSE.txt /user/hduser/wordcount/output
16/05/15 20:00:18 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
16/05/15 20:00:18 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
16/05/15 20:00:19 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
16/05/15 20:00:19 INFO input.FileInputFormat: Total input paths to process : 1
16/05/15 20:00:19 INFO mapreduce.JobSubmitter: number of splits:1
16/05/15 20:00:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local2102721529_0001
16/05/15 20:00:20 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
16/05/15 20:00:20 INFO mapreduce.Job: Running job: job_local2102721529_0001
16/05/15 20:00:20 INFO mapred.LocalJobRunner: OutputCommitter set in config null
16/05/15 20:00:20 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
```

图 7-18　运行 WordCount 程序

7.6 查看运行结果

运行后，程序会将结果存储在/user/hduser/wordcount/output 目录中，参照下列步骤查看结果。

步骤 01 查看输出目录

在“终端”程序中输入下列命令，查看运行结果：

➢ **查看 HDFS 的目录**

```
hadoop fs -ls /user/hduser/wordcount/output
```

屏幕显示界面如图 7-19 所示，生成了两个文件。

- _SUCCESS：代表程序运行成功。
- part-r-00000：运行结果的文本文件。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop fs -ls /user/hduser/wordcount/output
Found 2 items
-rw-r--r--   3 hduser supergroup          0 2016-05-15 20:00 /user/hduser/wordcount/output/_SUCCESS
-rw-r--r--   3 hduser supergroup       8006 2016-05-15 20:00 /user/hduser/wordcount/output/part-r-00000
hduser@master:~/wordcount$
```

图 7-19　查看输出目录

步骤 02 查看输出文件

在“终端”程序中输入下列命令，查看输出文件：

➢ **查看 HDFS 中的输出文件内容**

```
hadoop fs -cat /user/hduser/wordcount/output/part-r-00000|more
```

输出结果如图 7-20 所示，会列出每个英文单词出现的次数。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop fs -cat /user/hduser/wordcount/output/part-r-00000|more
"AS     4
"Contribution"  1
"Contributor"   1
"Derivative     1
"Legal  1
"License"       1
"License");     1
"Licensor"      1
"NOTICE"        1
"Not    1
"Object"        1
"Source"        1
"Work"  1
"You"   1
"Your") 1
"[]"    1
"control"       1
"printed        1
"submitted"     1
(50%)   1
(C)     1
(Don't  1
(INCLUDING      2
```

图 7-20　查看输出文件

步骤 03　删除输出目录

如果要再次执行 WordCount 程序，请先删除输出目录，这样才不会出现文件已经存在的报错信息。

```
hadoop fs -rm -R /user/hduser/wordcount/output
```

运行后的显示结果如图 7-21 所示。

```
hduser@master: ~/wordcount
hduser@master:~/wordcount$ hadoop fs -ls /user/hduser/wordcount/output
Found 2 items
-rw-r--r--   3 hduser supergroup          0 2016-05-15 20:00 /user/hduser/wordcount/output/_SUCCESS
-rw-r--r--   3 hduser supergroup       8006 2016-05-15 20:00 /user/hduser/wordcount/output/part-r-00000
hduser@master:~/wordcount$
```

图 7-21　删除输出目录

7.7 结论

本章我们介绍了 Hadoop MapReduce，同时发现它有以下缺点：

- 程序设计模式不容易使用，而且 Hadoop 的 Map Reduce API 太过低级，很难提高开发者的效率。
- 有运行效率的问题，MapReduce 需要将中间产生的数据保存到硬盘中，因此会有读写数据延迟的问题。
- 不支持实时处理，它原始的设计就是以批处理为主。

这些缺点可在下一章介绍的 Spark 中解决。

第 8 章

Python Spark的介绍与安装

本章将介绍 Spark 2.0 的安装，以及在 pyspark“终端”程序界面执行 Python Spark 程序于本机、Hadoop YARN-client 与 Spark Stand Alone 模式。

在 Spark 官网文件中，可以看到 Spark 的 Cluster 模式架构图（见图 8-1）。在浏览器输入下列官网网址：

```
http://spark.apache.org/docs/latest/cluster-overview.html
```

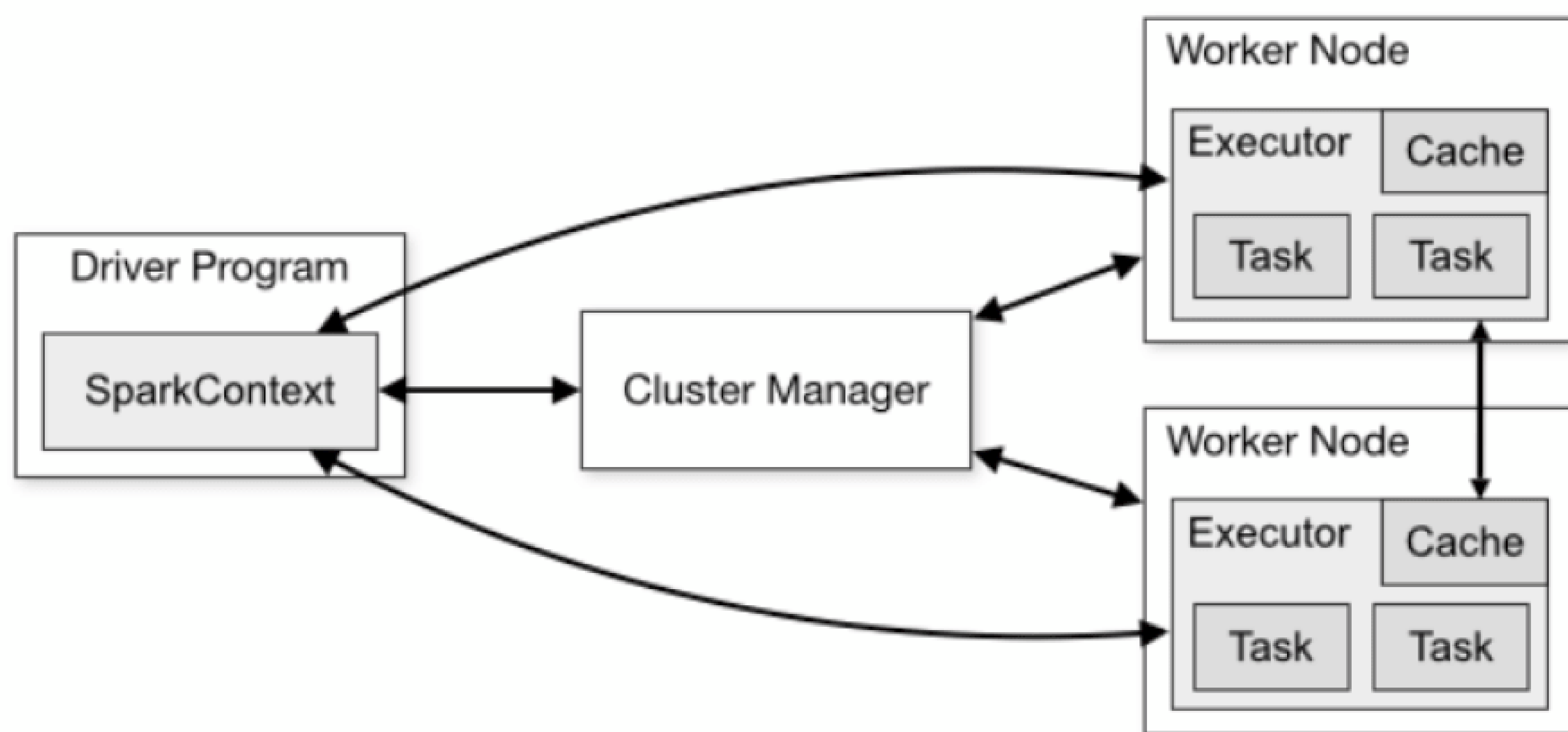


图 8-1　Spark 的 Cluster 模式架构图

参考图 8-1：

- DriverProgram 就是程序员所设计的 Spark 程序，在 Spark 程序中必须定义 SparkContext，它是开发 Spark 应用程序的入口。
- SparkContext 通过 Cluster Manager 管理整个集群，集群中包含多个 Worker Node，在每个 Worker Node 中都有 Executor 负责执行任务。

SparkContext 通过 Cluster Manager 管理整个集群 Cluster 的好处是：所设计的 Spark 程序可以在不同的 Cluster 模式下运行。

➢ **Cluster Manager 可以在下列模式下运行**

- 本地运行（Local Machine）——只需要在程序中 import Spark 的链接库就可以实现。在本地运行时，对于只安装在一台计算机上时最方便，适合刚入门时学习、测试使用。
- Spark Standalone Cluster——由 Spark 提供的 Cluster 管理模式，若没有架设 Hadoop Multi Node Cluster，则可架设 Spark Standalone Cluster，实现多台计算机并行计算。在这个模式下仍然可以直接存取 Local Disk 或是 HDFS。
- Hadoop YARN（Yet Another Resource Negotiator）——Hadoop 2.x 新架构中更高效率的资源管理核心。Spark 可以在 YARN 上运行，让 YARN 帮助它进行多台机器的资源管理。
- 在云端运行——针对更大型规模的计算工作，本地机器或自建集群的计算能力恐怕

难以满足。此时可以将 Spark 程序在云平台上运行，例如 AWS 的 EC2 平台。使用云计算的好处是不需要自己架设的费用，用多少付多少，随时可扩充。

本章将介绍如何在本地（Local Machine）、Spark Standalone Cluster、Hadoop YARN 运行 pyspark。

➢ **Spark 安装命令整理**

本章命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可节省打字的时间，也不用担心打错字（无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

8.1 Scala 的介绍与安装

Spark 支持 Scala、Java 和 Python 等语言，不过 Spark 本身是用 Scala 语言开发的，所以必须要先安装 Scala。Scala 具有以下特点：

- Scala 可编译为 Java bytecode 字节码，也就是说它可以在 JVM（Java Virtual Machine）上运行，具备跨平台能力。
- 现有 Java 的链接库都可以使用，可以继续使用丰富的 Java 开放源码生态系统。
- Scala 是一种函数式语言。在函数式语言中，函数也是值，与整数字符串等处于同一地位。函数可以作为参数传递给其他函数。
- Scala 是一种纯面向对象的语言，所有的东西都是对象，而所有的操作都是方法。

因为 Spark 本身是以 Scala 开发的，所以必须先安装 Scala。

步骤 01 下载 Scala 网址

我们将把 Scala 安装在 master 虚拟机上，在这个 Scala 下载网页可以看到各种不同版本的 Scala：

```
http://www.scala-lang.org/files/archive/
```

在图 8-2 所示的这个网页可以看到各种不同版本的 Scala，因为 Spark 2.0 必须配合 Scala 2.11 版本，在此我们选择 2.11.6 版本。

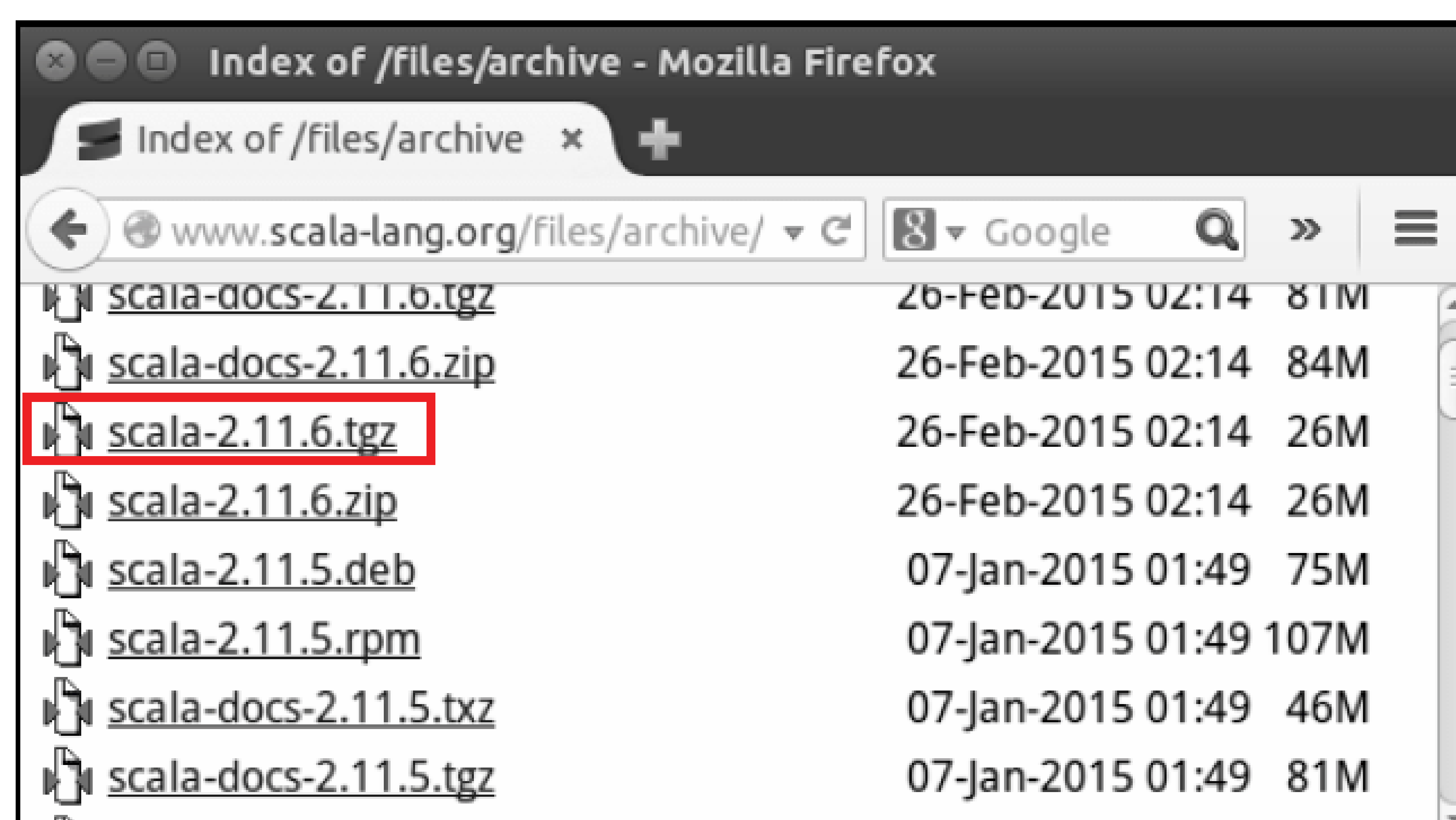


图 8-2　选择版本

步骤 02　下载 Scala

在本书中使用的是 scala-2.11.6 版本，所以在 master 虚拟机的“终端”程序中输入下列命令：

➢ **下载 scala-2.11.6 版本**

```
wget http://www.scala-lang.org/files/archive/scala-2.11.6.tgz
```

运行后屏幕显示界面如图 8-3 所示。



图 8-3　下载 Scala

步骤 03　解压缩 Scala

在“终端”程序中输入下列命令：

➢ **解压缩 scala-2.11.6.tgz 到 scala-2.11.6 目录**

```
tar xvf scala-2.11.6.tgz
```

运行后屏幕显示界面如图 8-4 所示。



图 8-4　解压缩 Scala

步骤04 把 Scala 迁移到/usr/local 目录

在“终端”程序中输入下列命令：

➢ **把 scala-2.11.6 迁移到/usr/local 目录**

```
sudo mv scala-2.11.6 /usr/local/scala
```

运行后屏幕显示界面如图 8-5 所示。



图 8-5　把 Scala 迁移到 /usr/local 目录

步骤05 Scala 用户环境变量的设置

在“终端”程序中输入下列命令：

➢ **启动 gedit 编辑~/.bashrc**

```
sudo gedit ~/.bashrc
```

按 Enter 键之后会启动 gedit 来编辑~/.bashrc，输入如图 8-6 所示的内容。

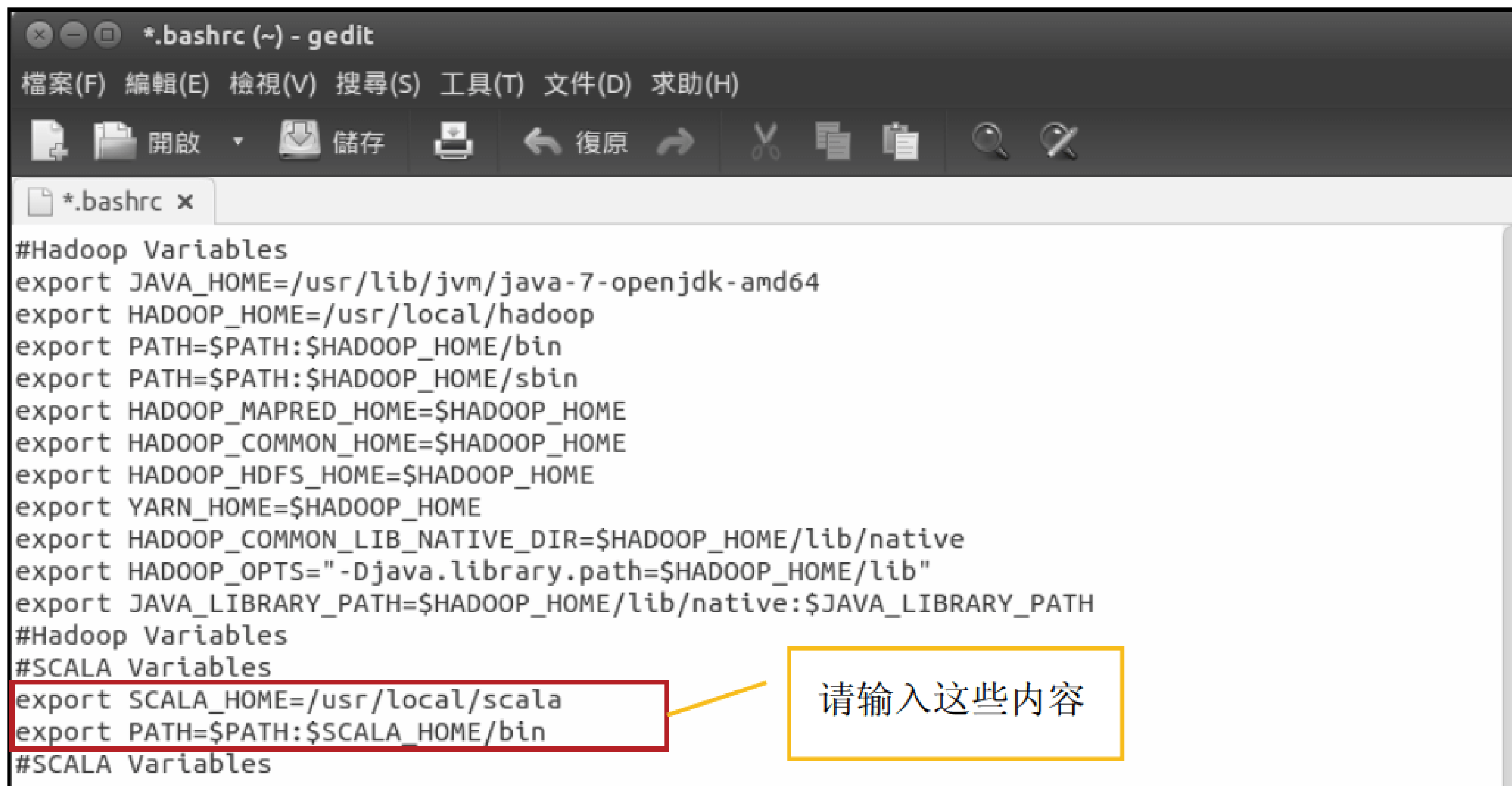


图 8-6　编辑 ~/.bashrc

说明如下：

➢ **设置 SCALA_HOME（scala 安装目录）为/usr/local/scala**

```
export SCALA_HOME=/usr/local/scala
```

➤ **设置 PATH 环境变量（加入$SCALA_HOME/bin）**

让我们在不同的目录下都可以执行 scala 程序。

```
export PATH=$PATH:$SCALA_HOME/bin
```

步骤 06 使~/.bashrc 修改生效

可以从系统注销再重新登录，或使用 source ~/.bashrc 命令让用户环境变量的设置生效，如图 8-7 所示。

```
source ~/.bashrc
```

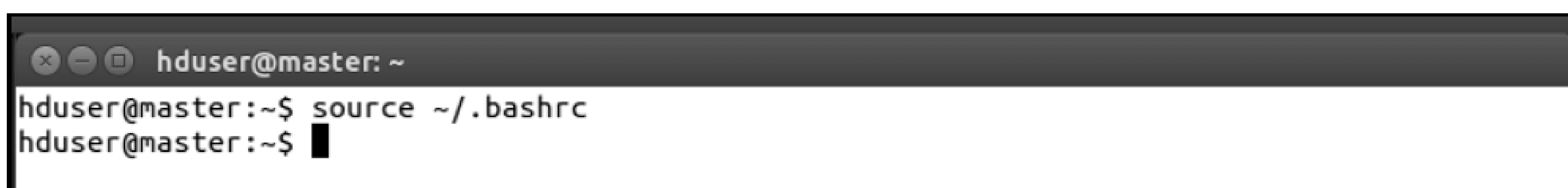


图 8-7　使用 ~/.bashrc 修改生效

步骤 07 启动 scala

至此，scala 已安装完成，接下来参照图 8-8 的说明进入 scala 交互式界面。

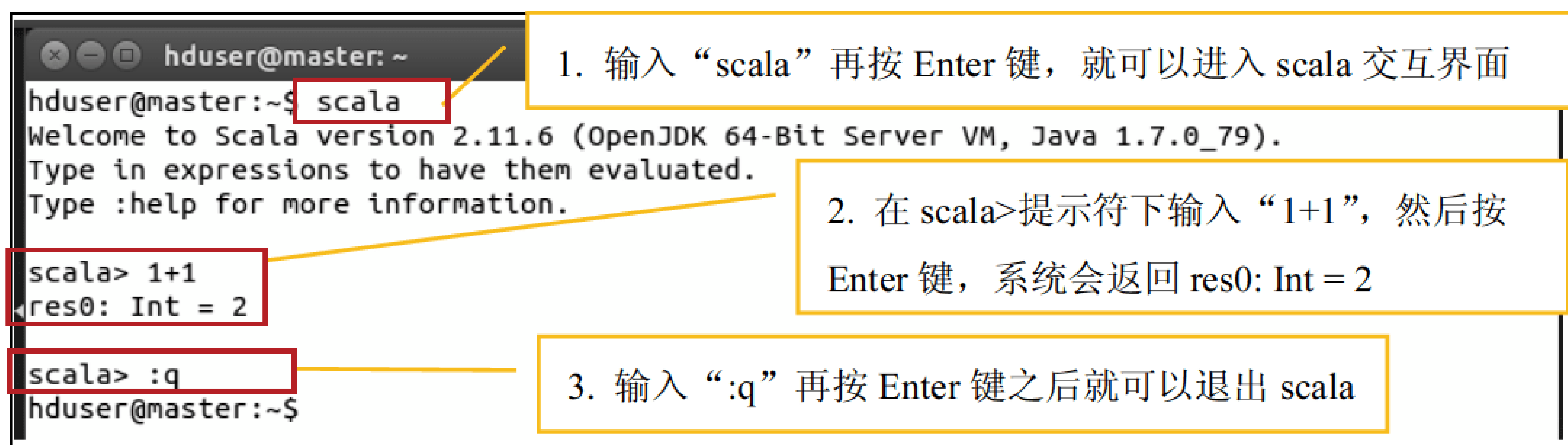


图 8-8　启动 scala

8.2 安装 Spark

接下来介绍如何安装 Spark。

步骤 01 Spark 下载页面

在浏览器输入下列网址来下载 Spark：

```
https://spark.apache.org/downloads.html
```

注意，在此选择“choose a package type”时，Spark 与 Hadoop 版本必须相互配合。因为

Spark 会读取 Hadoop HDFS 并且必须能在 HadoopYARN 执行程序，所以必须按照我们目前安装的 Hadoop 版本来选择 package type。因为我们之前安装的是 Hadoop 2.6.4，所以在此选择 Pre-built for Hadoop 2.6 and later，如图 8-9 所示。

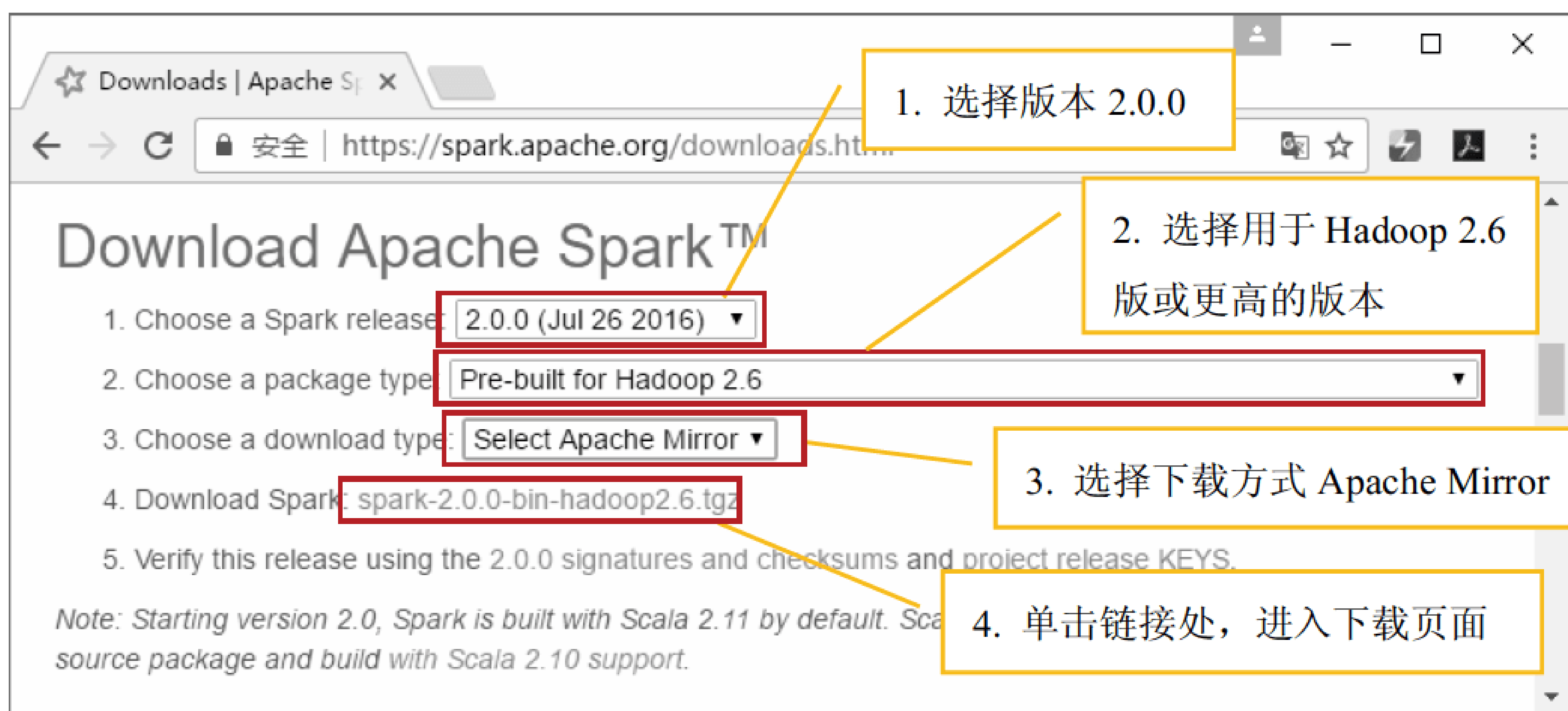


图 8-9　选择下载合适的 Spark 版本

步骤 02 打开 Spark 下载页面并复制链接

单击链接后，打开下载网页，我们可以选择离自己比较近的网络链接，复制链接的网址，如图 8-10 所示。

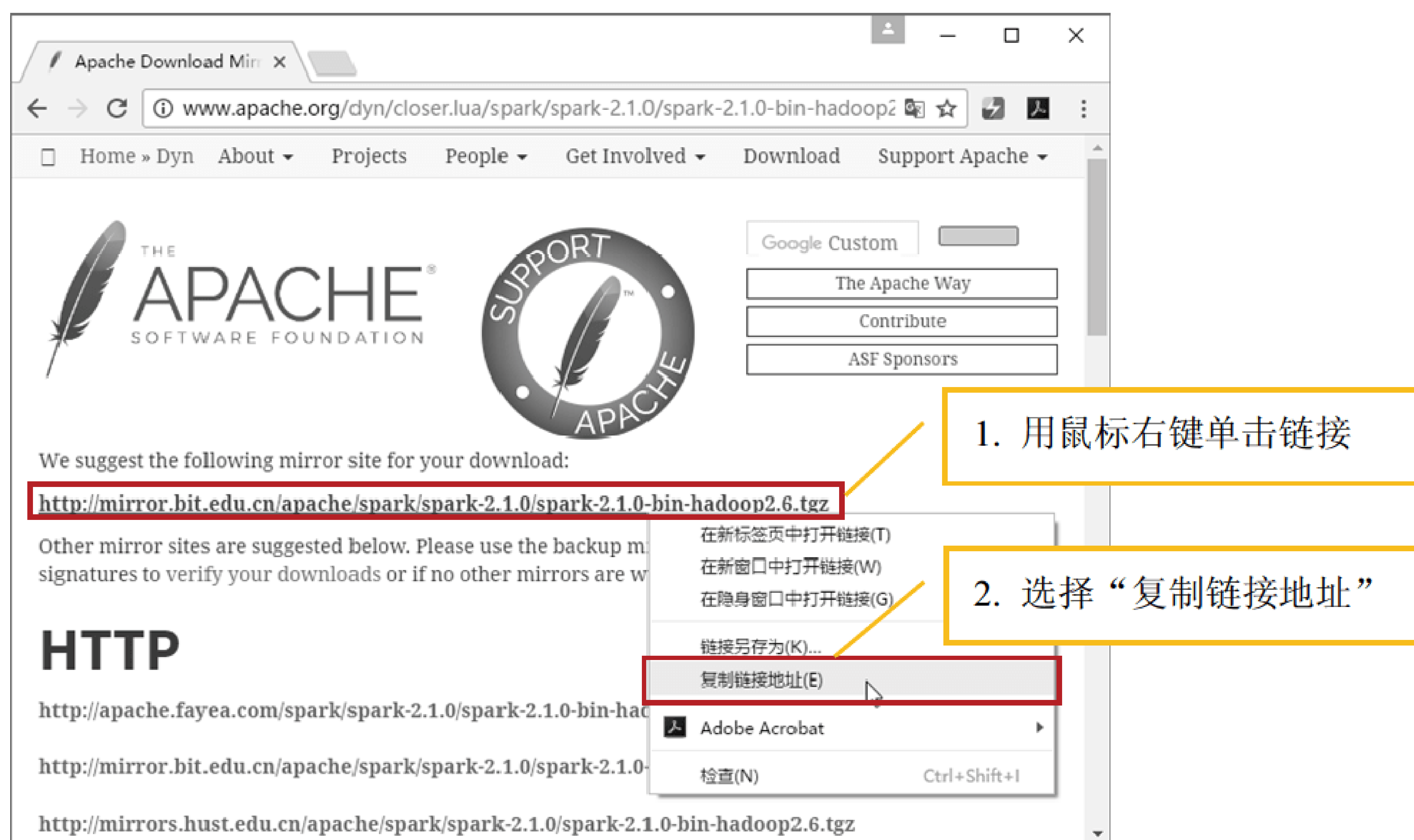


图 8-10　复制链接地址

步骤 03 安装 Spark

在“终端”程序中输入 wget，再按空格键，然后按 Ctrl + Shift + V 组合键粘贴之前所复

制的网址。

➢ **下载 Spark**

```
wget http://d3kbcqa49mib13.cloudfront.net/spark-2.0.0-bin-hadoop2.6.tgz
```

➢ **解压缩 Spark 到 spark-2.0.0-bin-hadoop2.6 目录**

```
tar zxf spark-2.0.0-bin-hadoop2.6.tgz
```

➢ **把 spark-2.0.0-bin-hadoop2.6 目录移动到/usr/local/spark**

```
sudo mv spark-2.0.0-bin-hadoop2.6 /usr/local/spark/
```

运行后屏幕显示界面如图 8-11 所示。

```
hduser@master: ~
hduser@master:~$ wget http://apache.stu.edu.tw/spark/spark-2.0.0/spark-2.0.0-bin
-hadoop2.6.tgz
--2016-08-04 06:50:59--  http://apache.stu.edu.tw/spark/spark-2.0.0/spark-2.0.0-
bin-hadoop2.6.tgz
正在解析主机 apache.stu.edu.tw (apache.stu.edu.tw)... 120.119.118.1, 2001:e10:c4
1:eeee::1
正在连接 apache.stu.edu.tw (apache.stu.edu.tw)|120.119.118.1|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 184354523 (176M) [application/x-gzip]
Saving to: 'spark-2.0.0-bin-hadoop2.6.tgz'

100%[======================================>] 184,354,523 1.22MB/s   in 1m 58s

2016-08-04 06:52:58 (1.49 MB/s) - 'spark-2.0.0-bin-hadoop2.6.tgz' saved [1843545
23/184354523]

hduser@master:~$ tar zxf spark-2.0.0-bin-hadoop2.6.tgz
hduser@master:~$ sudo mv spark-2.0.0-bin-hadoop2.6  /usr/local/spark/
```

图 8-11　安装 Spark

步骤 04 用户环境变量设置~/.bashrc

在“终端”程序中输入下列命令：

➢ **编辑~/.bashrc**

```
sudo gedit ~/.bashrc
```

按 Enter 键之后会打开~/.bashrc ，加入下列 Spark 环境变量（参考图 8-12）。



图 8-12　设置用户环境变量

上述命令说明如下：

➢ **设置 SPARK_HOME 为 Spark 安装目录/user/local/spark**

```
export SPARK_HOME=/usr/local/spark
```

➢ **设置 PATH 环境变量，加入$SPARK_HOME/bin**

这样在我们切换到不同的目录时仍可以执行 pyspark 程序。

```
export PATH=$PATH:$SPARK_HOME/bin
```

步骤05 *让用户环境变量的设置生效*

可以先注销再重新登录系统，让~/.bashrc 中的用户环境变量设置生效：

```
source ~/.bashrc
```

运行后屏幕显示界面如图 8-13 所示。

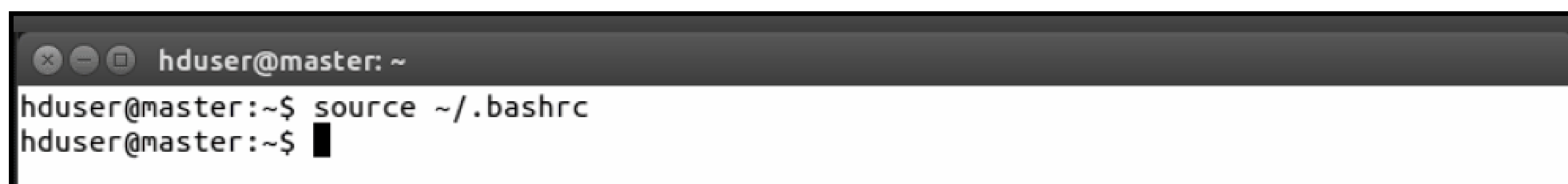


图 8-13 让用户环境变量设置生效

8.3 启动 pyspark 交互式界面

pyspark 的优点是具有交互的好处，输入命令立刻就可以看到响应。

步骤01 *启动 pyspark*

在“终端”程序中输入下列命令：

➢ **进入 Spark 交互式界面**

```
pyspark
```

运行后屏幕显示界面如图 8-14 所示，从中可以看到 Spark 与 Scala 的版本。

```
hduser@master:~$ pyspark
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
16/07/28 23:09:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)
SparkSession available as 'spark'.
>>>
```

Spark 版本

Scala 版本

最后会出现 scala>提示符，可以输入 Python 命令

图 8-14　启动 pyspark

步骤 02　退出 pyspark（见图 8-15）

请在 >>> 提示符号后输入下列命令：

➢ **离开 pyspark 交互式界面**

```
Exit()
```

```
>>>
>>>
>>>
>>>
>>>
>>> exit()
hduser@master:~$
```

图 8-15　退出 pyspark

8.4　设置 pyspark 显示信息

Spark 系统安装后，在 pyspark 交互式界面中默认会显示很多信息，有时太多信息会影响我们的阅读，所以建议修改设置，让它只显示警告信息。

步骤 01　复制 log4j 模板文件

在“终端”程序中输入下列命令：

➢ **切换到 spark 配置文件目录**

```
cd /usr/local/spark/conf
```

➢ **复制 log4j 模板文件到 log4j.properties**

```
cp log4j.properties.template log4j.properties
```

运行后屏幕显示界面如图 8-16 所示。

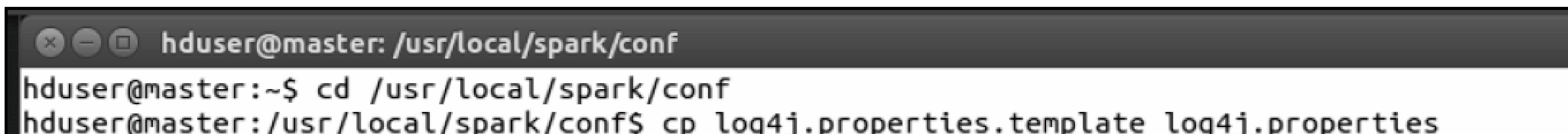


图 8-16　复制 log4j 模板文件

步骤 02 设置 log4j

在“终端”程序中输入下列命令，编辑 log4j.properties：

```
sudo gedit log4j.properties
```

按 Enter 键之后会打开 log4j.properties，屏幕显示界面如图 8-17 所示。

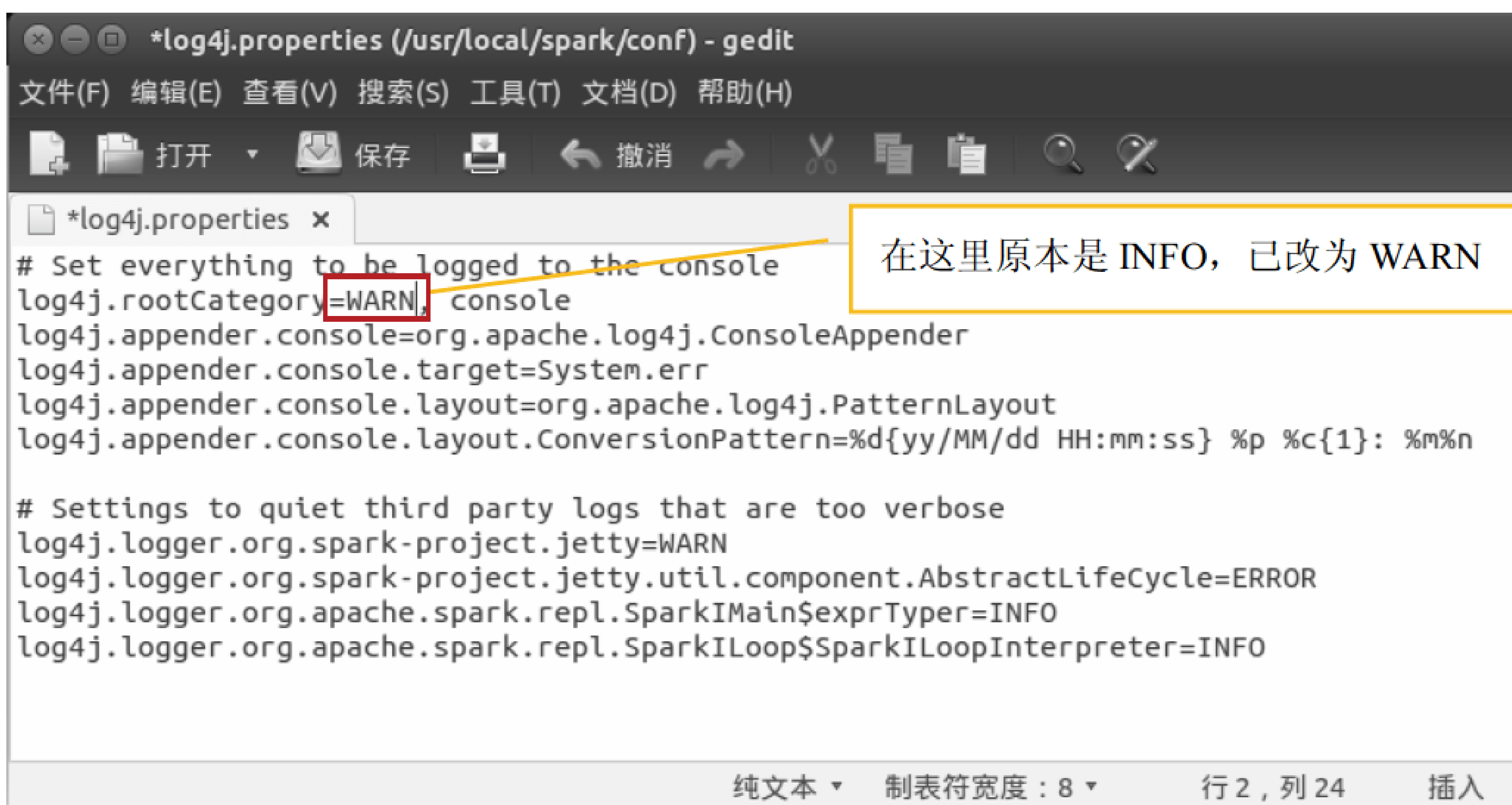


图 8-17　设置 log4j

参照图 8-17 将 log4j.rootCategory 原本的 INFO（信息）改为 WARN（警告），然后单击“保存”按钮关闭窗口。

步骤 03 再次进入 pyspark

再次在“终端”程序中输入下列命令，启动 pyspark：

```
pyspark
```

你会发现信息少了很多，只列出了重要部分的信息，如图 8-18 所示。

```
hduser@master:~$ pyspark
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
16/07/28 23:27:00 WARN NativeCodeLoader: Unable to load native-hadoop librar
sing builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)
SparkSession available as 'spark'.
>>>
```

图 8-18　再次进入 pyspark 显示的信息

8.5 创建测试用的文本文件

因为我们后续要测试 pyspark 读取 HDFS，所以必须先启动 Hadoop，并且上传文件到 HDFS。此部分如果在 7.4 节已经执行，就可以省略了。

步骤 01 复制 LICENSE.txt（见图 8-19）

在“终端”程序输入下列命令，复制文本文件：

➤ **复制** LICENSE.txt

```
cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
ll ~/wordcount/input
```

```
hduser@master:~$ cp /usr/local/hadoop/LICENSE.txt ~/wordcount/input
hduser@master:~$ ll ~/wordcount/input
总用量 24
drwxrwxr-x 2 hduser hduser  4096  5月 15 20:37 ./
drwxrwxr-x 3 hduser hduser  4096  5月 15 20:36 ../
-rw-r--r-- 1 hduser hduser 15429  5月 15 20:37 LICENSE.txt
hduser@master:~$
```

图 8-19　复制 LICENSE.txt

步骤 02 启动所有虚拟服务器

首先，必须要启动所有虚拟服务器，即 master、data1、data2、data3，如图 8-20 所示。



图 8-20 启动所有虚拟服务器

步骤 03 进入 master 虚拟机，启动 Hadoop Multi-Node Cluster

在 master 虚拟机，启动“终端”程序，输入下列指令：

➤ **启动 Hadoop Multi-Node Cluster**

```
start-all.sh
```

运行后屏幕显示界面如图 8-21 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namen
ode-master.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data2.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datano
de-data3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hd
user-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource
manager-master.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noden
anager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noden
anager-data3.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-noden
anager-data1.out
hduser@master:~$
```

图 8-21 启动 Hadoop Multi-Node Cluster

步骤 04 上传测试文件 HDFS 目录

接下来，我们将之前下载的文本文件从本机复制到 HDFS。

在“终端”程序输入下列命令：

➢ **在 HDFS 创建目录**

```
hadoop fs -mkdir -p /user/hduser/wordcount/input
```

➢ **切换至 ~/wordcount/input 数据文件目录**

```
cd ~/wordcount/input
```

➢ **上传文本文件到 HDFS**

```
hadoop fs -copyFromLocal LICENSE.txt /user/hduser/wordcount/input
```

➢ **列出 HDFS 文件**

```
hadoop fs -ls /user/hduser/wordcount/input
```

执行的结果如图 8-22 所示，从中可以看出已将 LICENSE.txt 上传到 HDFS 的 /user/hduser/wordcount/input 目录，

```
hduser@master: ~/wordcount/input
hduser@master:~$ hadoop fs -mkdir -p /user/hduser/wordcount/input
hduser@master:~$ cd ~/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -copyFromLocal LICENSE.txt /user/hdus
er/wordcount/input
hduser@master:~/wordcount/input$ hadoop fs -ls /user/hduser/wordcount/input
Found 2 items
-rw-r--r--   3 hduser supergroup      15429 2016-05-15 19:57 /user/hduser/wordco
unt/input/LICENSE.txt
```

图 8-22　上传测试文件 HDFS 目录

8.6 本地运行 pyspark 程序

步骤 01 进入 pyspark

首先，运行 pyspark 的最简单方式是在本地运行，可以在“终端”程序中输入下列命令：

➢ **本地运行 pyspark 程序**

```
pyspark--master local[4]
```

local[*N*]代表在本地运行，使用 *N* 个线程（thread），也就是说可以同时执行 *N* 个程序。虽然是在本地运行，但是因为现在 CPU 大多是多个核心，所以使用多个线程仍然会加速执行。local[*] 会尽量使用我们机器上的 CPU 核心，我们也可以指定使用的线程数，例如 local[4] 代表使用 4 个线程（thread）。

运行后屏幕显示界面如图 8-23 所示。

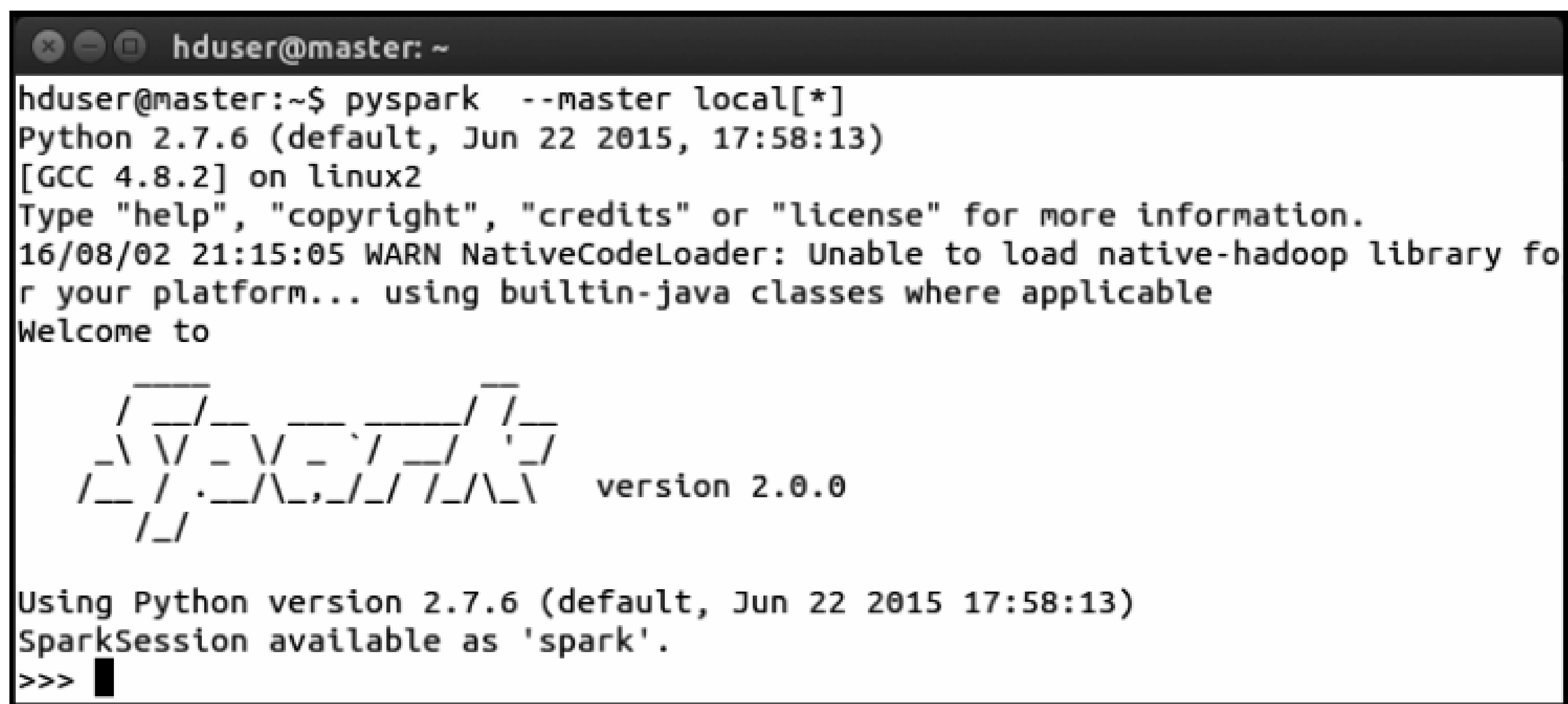


图 8-23 在本地运行 pyspark

步骤02 查看当前的运行模式

在 >>> 提示符号下输入下列命令，查看当前的运行模式：

```
sc.master
```

运行后屏幕显示界面如图 8-24 所示，我们可以看到返回了 local[*]。

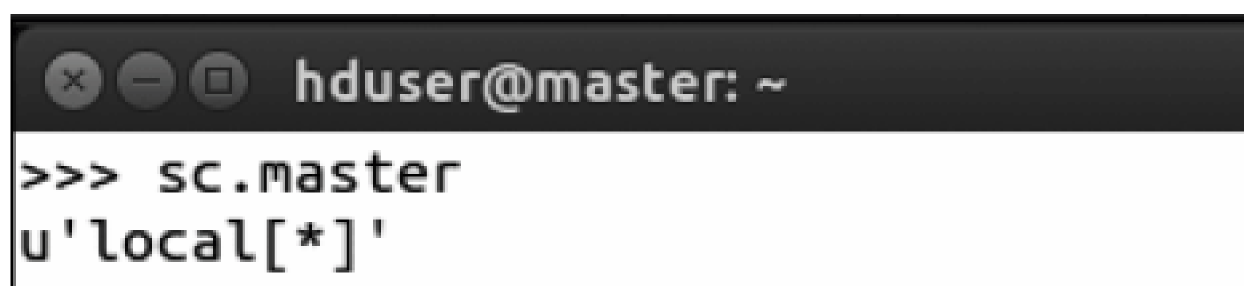


图 8-24 查看当前的运行模式

步骤03 读取本地文件

我们将执行一个简单的 Spark 程序，在>>>提示符后输入下列命令：

➢ 读取本地的文件

路径前加上“file:”，告诉系统要读取 HDFS 文件。

```
textFile=sc.textFile("file:/usr/local/spark/README.md")
```

➢ 显示项数

```
textFile.count()
```

运行后屏幕显示界面如图 8-25 所示，我们可以看到共有 99 项数据。



图 8-25 读取本地文件

步骤04 读取 HDFS 文件

请在 >>> 提示符号后输入下列命令：

➢ **读取 HDFS 文件**

在路径前加上“hdfs://master:9000”，告诉系统要读取 HDFS 文件。

```
textFile =sc.textFile("hdfs://master:9000/user/hduser/wordcount/
input/LICENSE.txt")
```

运行后屏幕显示界面如图 8-26 所示。

```
hduser@master: ~
>>>
>>>
>>> textFile =sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/LICENSE.txt")
>>> textFile.count()
289
>>>
>>>
```

图 8-26　读取 HDFS 文件

以上 HDFS 存取路径为“hdfs://master:9000”，是我们在安装 Hadoop 时设置的，请参考第 5.3 节步骤 5“编辑 core-site.xml”。

步骤 05　退出 pyspark（见图 8-27）

请在 pyspark 提示符号 >>> 输入下列命令：

```
exit()
```

```
hduser@master: /usr/local/spark/conf
>>> exit()
hduser@master:/usr/local/spark/conf$
```

图 8-27　退出 pyspark

8.7 在 Hadoop YARN 运行 pyspark

Spark 可以在 Hadoop YARN 上运行，让 YARN 帮助它进行多台机器资源的管理。接下来介绍如何在 Hadoop YARN 运行 pyspark。

步骤 01　在 Hadoop YARN 运行 pyspark

在 master 虚拟机启动“终端”程序，输入下列命令：

➢ **在 Hadoop YARN 运行 pyspark**

```
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop pyspark --master yarn -deploy
-mode client
```

运行后，屏幕显示界面如图 8-28 所示，最后会出现>>>提示符。

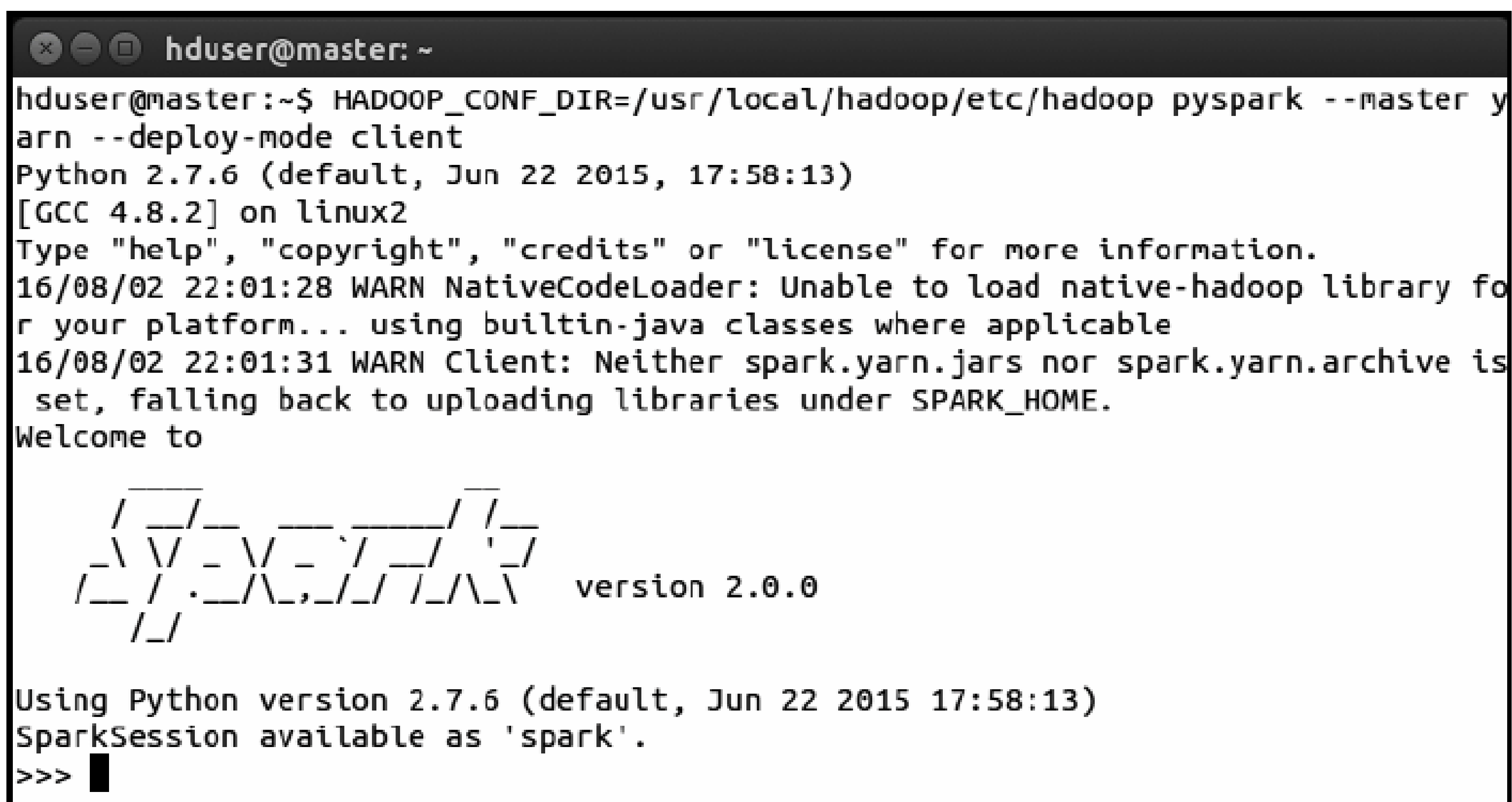


图 8-28　在 Hadoop YARN 运行 pyspark

以上命令详细说明如下：

➢ **设置 Hadoop 配置文件目录**

```
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

➢ **要运行的程序 pyspark**

```
pyspark
```

➢ **设置运行模式是 YARN-client**

```
--master yarn --deploy-mode client
```

步骤 02　查看当前的运行模式

在 >>> 提示符号下输入下列命令：

➢ **查看当前的运行模式**

```
Sc.master
```

运行结果如图 8-29 所示，可以看到返回了 YARN-client。

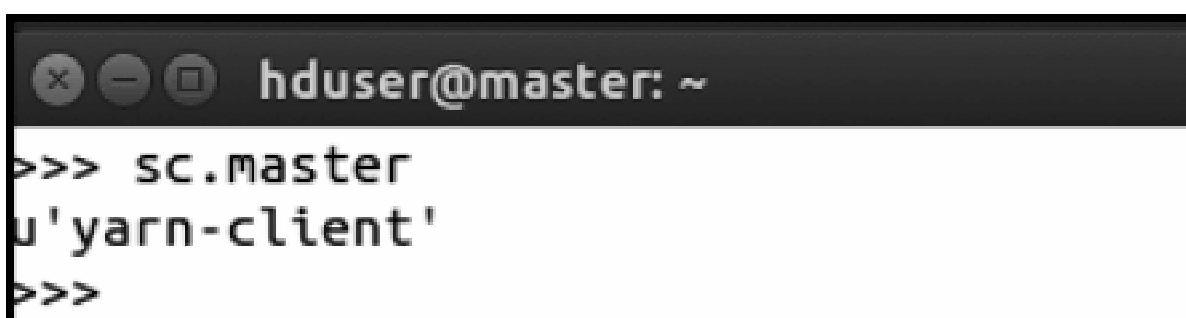


图 8-29　查看当前的运行模式

步骤 03　读取 HDFS 文件

在>>>提示符下输入下列命令，读取 HDFS 文件并显示笔数：

```
textFile= sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/
LICENSE.txt")
```

运行结果如图 8-30 所示。

```
hduser@master: ~
>>> textFile =sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/LICENSE.txt")
>>> textFile.count()
289
>>> 
```

图 8-30　读取 HDFS 文件

步骤 04　在 Hadoop Web 界面查看 PySparkShell App

现在我们已经在 Hadoop YARN 运行了 pyspark，所以在 Hadoop Web 界面就可以看到这个应用程序（Application）了。

```
URL：http://localhost:8088/
```

参照图 8-31 所示的步骤打开 Hadoop Web 界面。



图 8-31　在 Hadoop Web 界面查看 PySparkShell App

8.8　构建 Spark Standalone Cluster 运行环境

接下来我们将构建 Spark Standalone Cluster 运行环境，然后测试 pyspark 在 Spark Standalone Cluster 环境中的运行。具体步骤如表 8-1 所示。

表 8-1　构建 Spark Standalone Cluster 运行环境

构建步骤	说明
1. 在 master 虚拟机设置 spark-env.sh	设置 Spark Standalone Worker 使用的 CPU、内存等
2. 复制 spark 程序到 data1、data2、data3	将 master 的 spark 程序复制到 data1、data2、data3
3. 在 master 虚拟机编辑 slaves 文件	设置 Spark Standalone Cluster 有哪些服务器

步骤01 复制模板文件（template）来创建 spark-env.sh

spark-env.sh 是 Spark 的环境配置文件。Spark 系统中提供了模板文件，便于用户设置时作为参照。在 master 虚拟机中启动“终端”程序，输入下列命令：

➢ **复制模板文件来创建 spark-env.sh**

```
cp /usr/local/spark/conf/spark-env.sh.template /usr/local/spark/conf/
spark-env.sh
```

运行后，屏幕显示界面如图 8-32 所示。

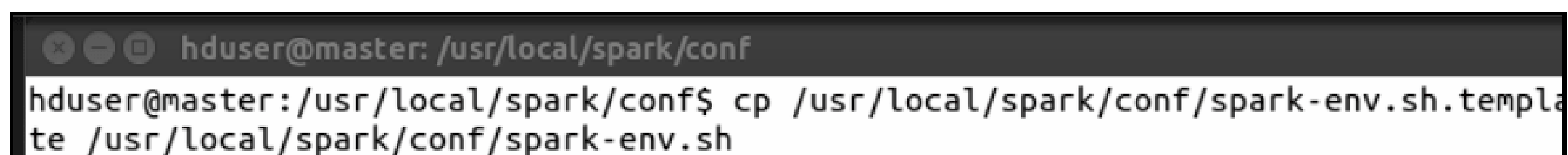


图 8-32　复制模板文件来创建 spark-env.sh

步骤02 设置 spark-env.sh

可以在“终端”程序输入下列命令：

➢ **编辑 spark-env.sh**

```
sudo gedit /usr/local/spark/conf/spark-env.sh
```

按 Enter 键之后就会用 gedit 打开 spark-env.sh 文件，如图 8-33 所示。

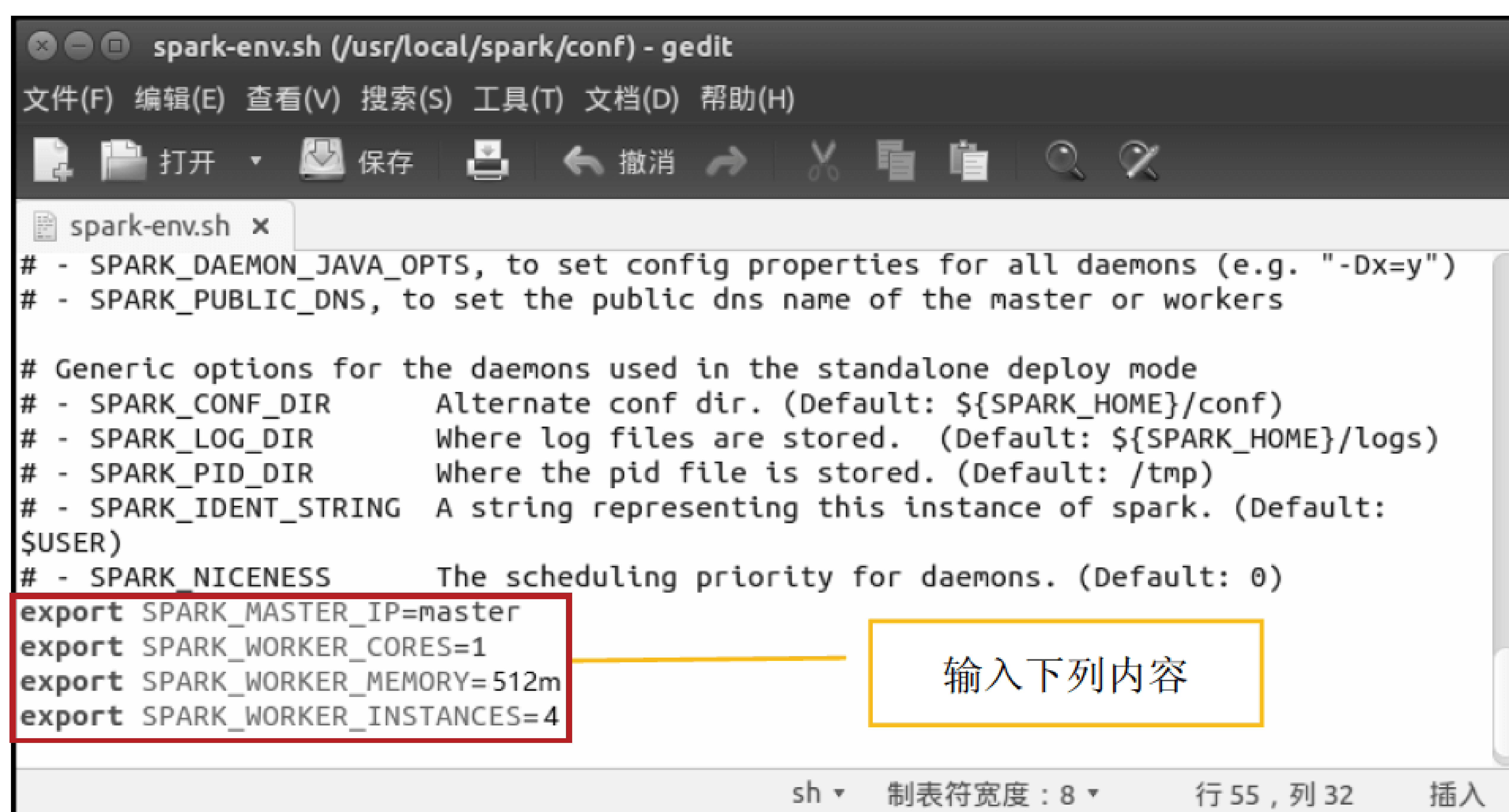


图 8-33　编辑 spark-env.sh

spark-env.sh 文件的设置内容，说明如下：

➢ **设置 master 的 IP 或服务器名称**

```
export SPARK_MASTER_IP=master
```

➢ **设置每个 Worker 使用的 CPU 核心**

```
export SPARK_WORKER_CORES=1
```

➢ **设置每个 Worker 使用的内存**

```
export SPARK_WORKER_MEMORY=512m
```

内存的设置必须考虑所使用计算机的内存容量，如果计算机的容量不足，就不要设置太多。

➢ **设置实例数**

```
export SPARK_WORKER_INSTANCES=4
```

步骤 03 将 master 的 spark 程序复制到 data1

接下来，我们将 master 的 spark 程序复制到 data1，具体步骤如下：

- 首先在 master 的“终端”程序中使用 ssh 连接到 data1，创建目录、更改所有者。
- 然后使用 scp 将 master 的 spark 程序复制到 data1。

提示

scp（secure copy 的缩写）可在 Linux 加密传输、进行远程文件复制，以及从本地主机复制文件到远程主机。scp 命令的功能很多，不过在这里只介绍最基本的功能，命令格式如下：

```
scp  -r [本地文件]  [远程用户名称]@[远程主机名] : [远程目录]
```

-r　递归复制整个目录。

在“终端”程序中输入下列命令：

➢ **SSH 连接至 data1**

```
ssh data1
```

➢ **连接成功后创建 spark 目录**

```
sudo mkdir /usr/local/spark
```

➢ **更改所有者为 hduser**

```
sudo chown hduser:hduser /usr/local/spark
```

➢ **注销 data1**

```
exit
```

➢ **使用 scp 将 master 的 spark 程序复制到 data1**

```
sudo scp -r /usr/local/spark hduser@data1:/usr/local
```

运行后，屏幕显示界面如图 8-34 所示。

```
hduser@master: ~
hduser@master:~$ ssh data1
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

148 packages can be updated.
101 updates are security updates.

Last login: Sat Jun 25 17:43:43 2016 from master
hduser@data1:~$ sudo mkdir /usr/local/spark
[sudo] password for hduser:
hduser@data1:~$ sudo chown hduser:hduser /usr/local/spark
hduser@data1:~$ exit
logout
Connection to data1 closed.
hduser@master:~$ sudo scp -r /usr/local/spark hduser@data1:/usr/local
The authenticity of host 'data1 (192.168.56.101)' can't be established.
ECDSA key fingerprint is 67:00:38:8d:57:5e:64:58:d5:6a:92:8c:c7:65:22:8f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'data1,192.168.56.101' (ECDSA) to the list of known h
osts.
hduser@data1's password:
```

1. SSH 连接至 data1
2. 连接成功后创建 spark 目录
3. 更改所有者为 hduser
5. scp 从 master 复制到 data1
6. 输入 yes
7. 输入 data1 密码

图 8-34　使用 scp 复制 spark 程序到 data1

使用 scp 复制时，系统会询问是否继续，请输入“yes”。因为要复制文件到 data1，所以必须输入 data1 密码。

步骤 04 将 master 的 spark 程序复制到 data2

在“终端”程序中输入下列命令，通过 ssh 与 scp 将 master 的 spark 程序复制到 data2：

➢ **SSH 连接至 data2**

```
ssh data2
```

➢ **连接成功后创建 spark 目录**

```
sudo mkdir /usr/local/spark
```

➢ **更改所有者为 hduser**

```
sudo chown hduser:hduser /usr/local/spark
```

➢ **注销 data2**

```
exit
```

➢ **使用 scp 将 master 的 spark 程序复制到 data2**

使用 scp 复制时，系统会询问是否继续，请输入“yes”。因为要复制文件到 data2，所以必须输入 data2 密码。

```
sudo scp -r /usr/local/spark hduser@data2:/usr/local
```

运行后，屏幕显示界面如图 8-35 所示。

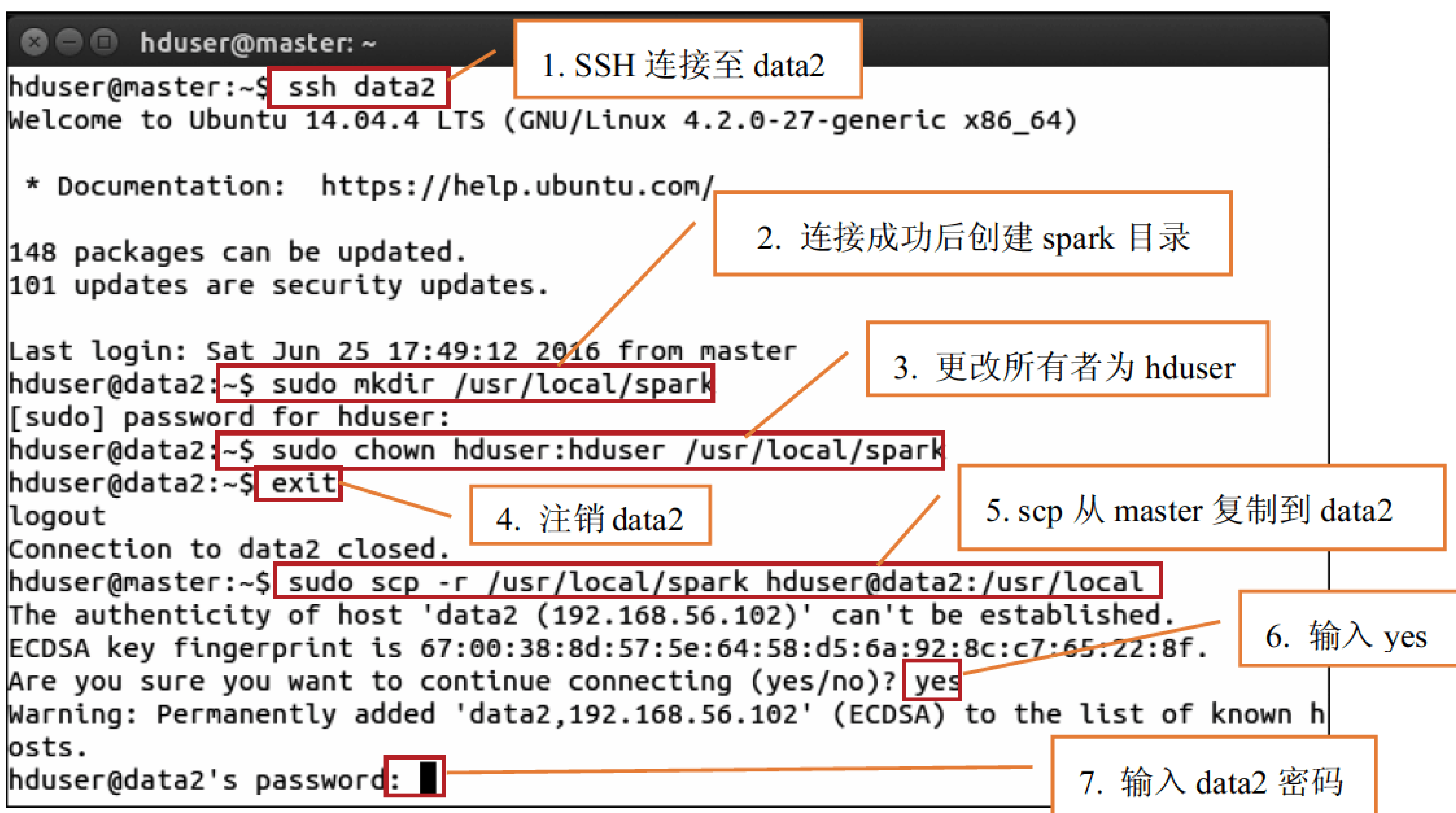


图 8-35　复制 spark 程序到 data2

步骤 05 将 master 的 spark 程序复制到 data3

在“终端”程序中输入下列命令，通过 ssh 与 scp 将 master 的 spark 程序复制到 data3：

➢ **SSH 连接至 data3**

```
ssh data3
```

➢ **连接成功后创建 spark 目录**

```
sudo mkdir /usr/local/spark
```

➢ **更改所有者为 hduser**

```
sudo chown hduser:hduser /usr/local/spark
```

➢ **注销 data3**

```
exit
```

➢ **使用 scp 将 master 的 spark 程序复制到 data3**

使用 scp 复制时，系统会询问是否继续，请输入“yes”。因为要复制文件到 data3，所以必须输入 data3 密码。

```
sudo scp -r /usr/local/spark hduser@data3:/usr/local
```

运行后，屏幕显示界面如图 8-36 所示。

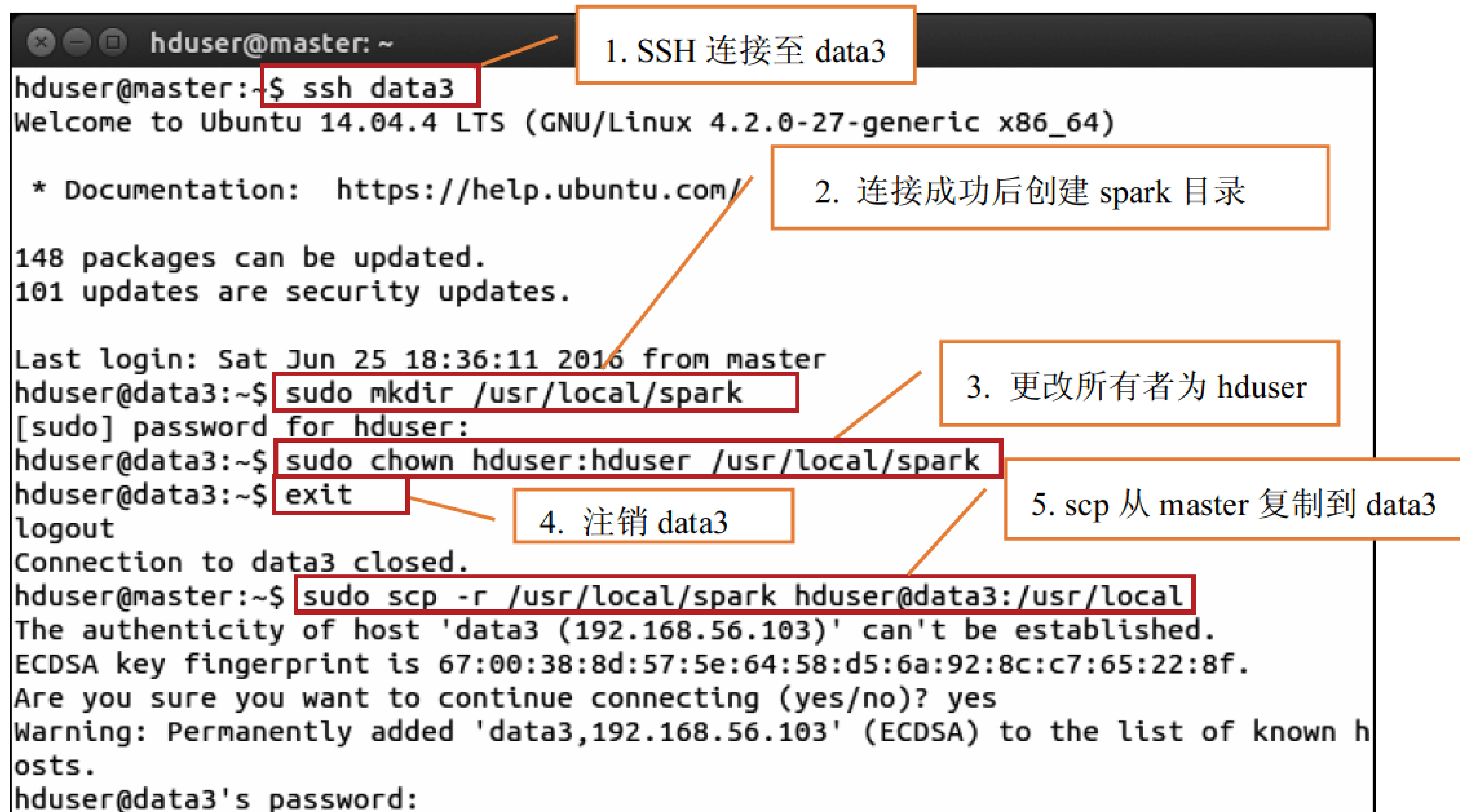


图 8-36　将 master 的 spark 程序复制到 data3

步骤 06 编辑 slaves 文件

可以在“终端”程序中输入下列命令：

➢ 编辑 slaves 文件

设置 Spark Standalone Cluster 有哪些服务器：

```
sudo gedit /usr/local/spark/conf/slaves
```

按 Enter 键之后就会用 gedit 打开 slaves 文件，如图 8-37 所示。



图 8-37　编辑 slaves 文件

8.9 在 Spark Standalone 运行 pyspark

步骤01 启动 Spark Standalone Cluster

可以在“终端”程序中输入下列命令：

➢ **启动 Spark Standalone Cluster**

```
/usr/local/spark/sbin/start-all.sh
```

运行后屏幕显示界面如图 8-38 所示，可以看到一共启动了 3 个 worker。

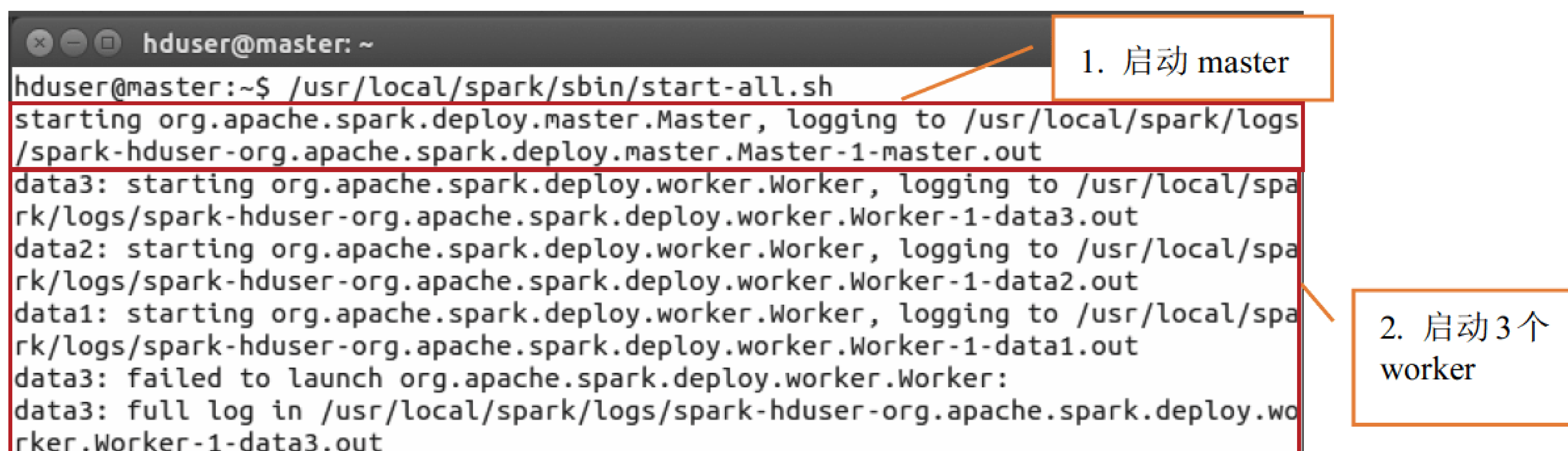


图 8-38 启动 Spark Standalone Cluster

➢ **分别启动 master 与 slaves**

之前我们使用 start-all.sh 会同时启动 master 与 slaves，也可以分别启动 master 与 slaves，命令如表 8-2 所示。

表 8-2 分别启动 master 与 slaves

命令	说明
/usr/local/spark/sbin/start-master.sh	启动 master 服务器
/usr/local/spark/sbin/start-slaves.sh	启动 slaves 服务器

步骤02 在 Spark Standalone 运行 pyspark

可以在“终端”程序中输入下列命令：

➢ **在 Spark Standalone 运行 pyspark 程序**

```
pyspark --master spark://master:7077 --num-executors 1 --total-executor
```

```
-cores 3
   --executor-memory 512m
```

运行后屏幕显示界面如图 8-39 所示。

```
hduser@master: ~
hduser@master:~$ pyspark --master spark://master:7077 --num-executors 1 --total-e
xecutor-cores 3 --executor-memory 512m
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
16/07/28 23:56:36 WARN NativeCodeLoader: Unable to load native-hadoop library for
 your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)
SparkSession available as 'spark'.
>>>
```

图 8-39　在 Spark Standalone 运行 pyspark

步骤 03 查看当前的运行模式

在 >>> 提示符号下输入下列命令：

➢ 查看当前的运行模式

```
sc.master
```

运行后屏幕显示界面如图 8-40 所示，我们可以看到返回了 spark://master:7077。

```
hduser@master: ~
>>> sc.master
u'spark://master:7077'
>>>
```

图 8-40　查看当前的运行模式

步骤 04 读取本地文件

在>>>提示符后输入下列命令读取本地文件：

```
textFile=sc.textFile("file:/usr/local/spark/README.md")
textFile.count
```

运行后，屏幕显示界面如图 8-41 所示。

```
hduser@master: ~
>>> textFile=sc.textFile("file:/usr/local/spark/README.md")
>>> textFile.count()
99
>>>
```

图 8-41　读取本地文件

注意，当在 cluster 模式（如 YARN-client 或 Spark stand alone 模式）读取本地文件时，因为程序会分布在不同的机器中执行，所以必须确认所有的机器都有该文件，否则会发生错误。

如上范例，我们读取 file:/usr/local/spark/README.md，因为每一个机器都有该文件，所以不会发生错误。建议最好在 cluster 模式读取 HDFS 文件，这样才不会有问题。

步骤 05 读取 HDFS 文件

在 >>> 提示符号输入下列命令，读取 HDFS 文件并显示笔数。

```
    textFile=sc.textFile("hdfs://master:9000/user/hduser/wordcount/input/
LICENSE.txt")
    textFile.count()
```

运行后屏幕显示界面如图 8-42 所示。

```
hduser@master: ~
>>> textFile =    sc.textFile("hdfs://master:9000/user/hduser/wordcount/input
/LICENSE.txt")
>>> textFile.count()
289
>>> 
```

图 8-42　读取 HDFS 文件

8.10 Spark Web UI 界面

现在我们已经在 Spark Standalone cluster 运行了 pyspark，所以可以在 Spark Web UI 界面看到这个应用程序（Application）。

步骤 01 启动 Spark Standalone Web UI 界面

可以在浏览器输入下列网址，查看 Spark Standalone Web UI 界面。

```
    http://master:8080/
```

运行后屏幕显示界面如图 8-43 所示，从中可以看到启动后共有 3 个 worker 与当前运行的程序 pyspark。

步骤 02 查看 Spark Jobs

单击 PySparkShell 链接后，我们可以看到 Spark Jobs 有两个 job，分别是之前读取本地与 HDFS 的 job，如图 8-44 所示。

Spark Master at spark://master:7077 - Mozilla Firefox

master:8080

Spark 2.0.0 Spark Master at spark://master:7077

URL: spark://master:7077
REST URL: spark://master:6066 *(cluster mode)*
Alive Workers: 3
Cores in use: 3 Total, 3 Used
Memory in use: 1536.0 MB Total, 1536.0 MB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

1．确认启动后共有 3 个 worker

Workers

Worker Id	Address	State	Cores	Memory
worker-20160729003131-192.168.56.101-43413	192.168.56.101:43413	ALIVE	1 (1 Used)	512.0 MB (512.0 MB Used)
worker-20160729003131-192.168.56.102-33018	192.168.56.102:33018	ALIVE	1 (1 Used)	512.0 MB (512.0 MB Used)
worker-20160729003132-192.168.56.103-44217	192.168.56.103:44217	ALIVE	1 (1 Used)	512.0 MB (512.0 MB Used)

2．单击此 PySparkShell 链接

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160729003219-0000 (kill)	PySparkShell	3	512.0 MB	2016/07/29 00:32:19	hduser	RUNNING	49 s

图 8-43 启动 Spark Standalone Web UI 界面

PySparkShell - Spark Jobs - Mozilla Firefox

192.168.56.100:4040/jobs/

Spark 1.6.1 Jobs Stages Storage Environment Executors PySparkShell application UI

Spark Jobs (?)

Total Uptime: 18 min
Scheduling Mode: FIFO
Completed Jobs: 2

Event Timeline

单击 Job Id 是 1 的链接

Completed Jobs (2)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	count at <stdin>:1	2016/07/25 17:15:48	5 s	1/1	2/2
0	count at <stdin>:1	2016/07/25 17:14:37	5 s	1/1	2/2

图 8-44 查看 Spark Jobs

步骤 03 查看 Spark Jobs1 详细信息

可以用鼠标单击 Job 来查看 Job 详细的执行过程，如图 8-45 所示，这些信息可用于我们调试或 Performance Tuning（性能调优）时找出那些执行缓慢的 Job。

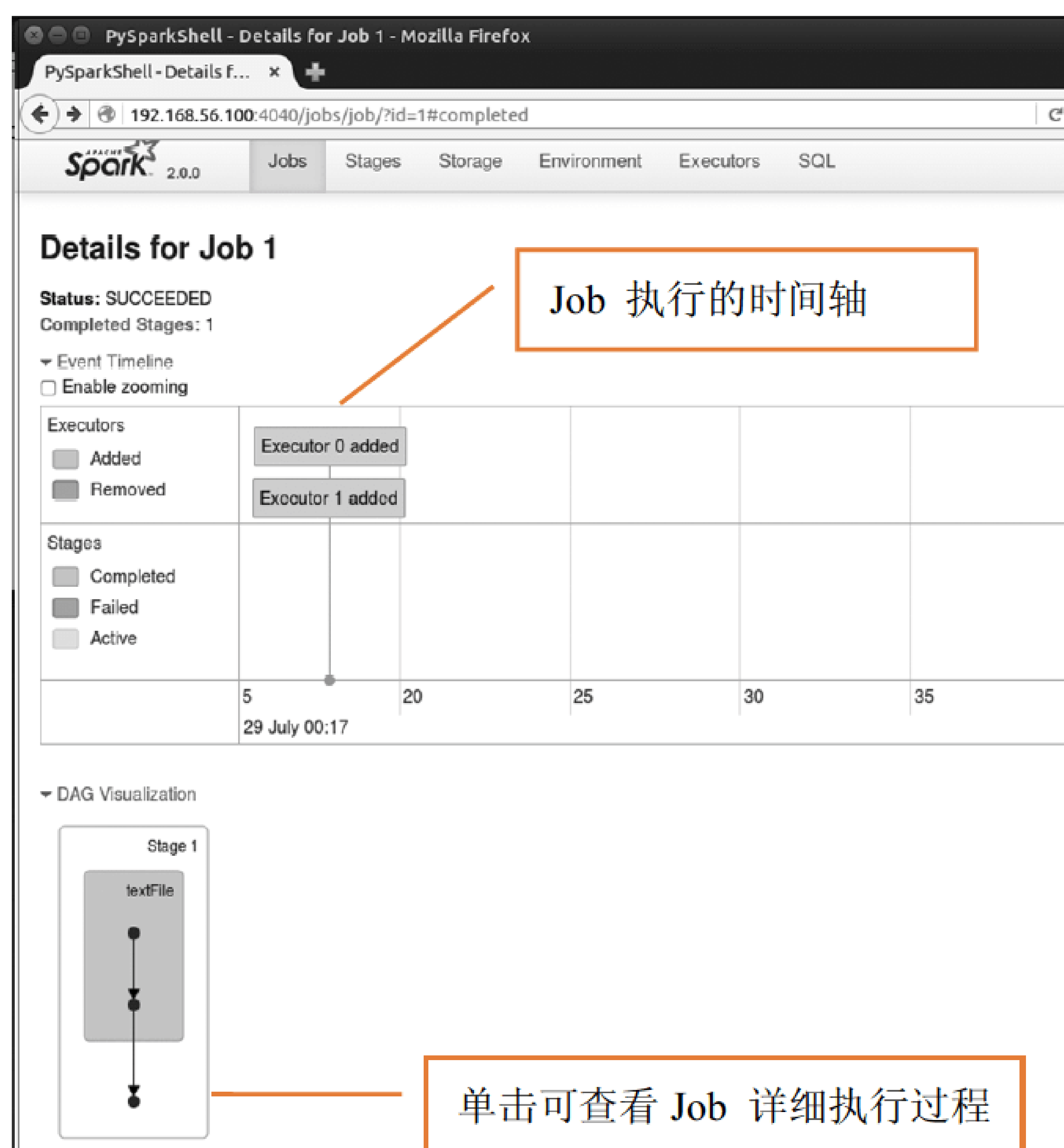


图 8-45　查看 Spark Jobs1 详细信息

步骤 04 停止 Spark standalone cluster

在“终端”程序中输入下列指令，停止 Spark standalone cluster：

```
/usr/local/spark/sbin/stop-all.sh
```

运行后，屏幕显示界面如图 8-46 所示。

```
hduser@master: ~
hduser@master:~$ /usr/local/spark/sbin/stop-all.sh
data2: stopping org.apache.spark.deploy.worker.Worker
data3: stopping org.apache.spark.deploy.worker.Worker
data1: stopping org.apache.spark.deploy.worker.Worker
stopping org.apache.spark.deploy.master.Master
hduser@master:~$
```

图 8-46　停止 Spark standalone cluster

8.11 结论

本章我们介绍了 Spark 的安装，并且示范了如何使用 Python Spark 在不同模式下执行、读取本地文件与 HDFS 文件。只是在 pyspark 是在“终端”程序中使用纯文本界面时较不方便，而且执行的命令与结果也无法记录，所以下一章我们将介绍 IPython Notebook。IPython Notebook 可以让我们将数据分析的过程、运行后的命令与结果存储成笔记本，下次打开笔记本时，就可以重新执行这些命令。

第 9 章

在IPython Notebook运行 Python Spark程序

安装 Python 最方便的方式是使用软件包来安装。安装 Anaconda 软件包时会同时安装很多软件包，包括 IPython Notebook、NumPy、SciPy、Matplotlib。这几个是用于数据分析、科学计算上的常用软件包。

IPython Notebook 具备交互式界面，我们可以在 Web 界面输入 Python 命令后立刻看到结果。我们还可将数据分析的过程和运行后的命令与结果存储成笔记本，下次可以打开笔记本，重新执行这些命令，IPython Notebook 笔记本可以包含文字、数学公式、程序代码、结果、图形、视频。因为 IPython Notebook 是功能强大的交互式界面，很适合数据分析，所以在后续章节中我们会使用 IPython Notebook 示范 Spark 的命令。

9.1 安装 Anaconda

安装 Anaconda 的步骤如下：

步骤 01 复制安装 Anaconda 的下载网址

➢ **连接 continuum 网址**

```
https://repo.continuum.io/archive/index.html
```

当向下浏览时，我们可以看到 Anaconda for Linux，这里选择下载 Anaconda 2-2.5.0-Linux-x86_64.sh（见图 9-1）。因为此版本与 Spark 2.0 兼容，所以建议安装此版本。

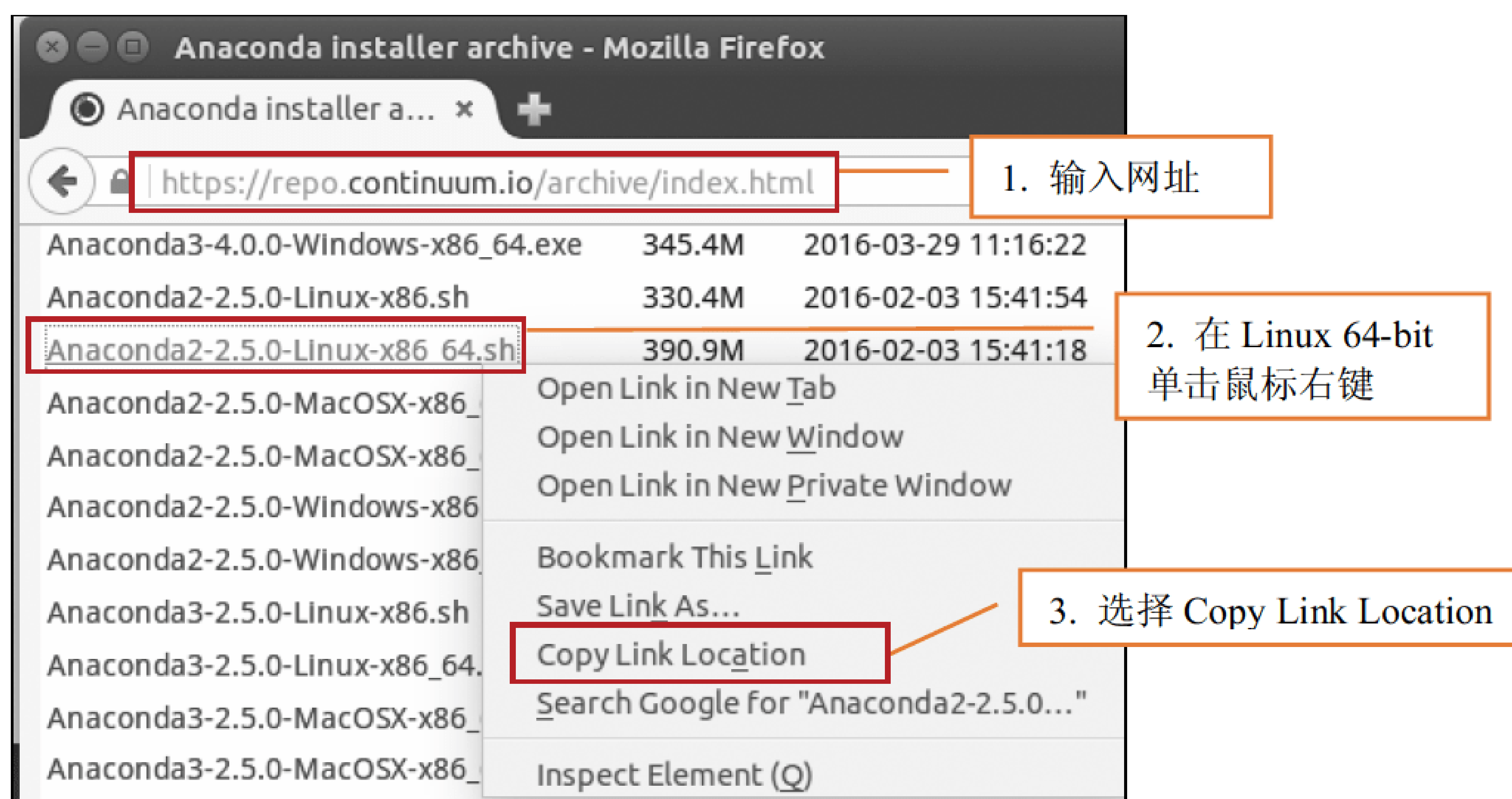


图 9-1 选择安装 Anaconda 的下载网址

步骤 02 下载 Anaconda2-2.5.0-Linux-x86_64.sh

在“终端”程序输入 wget 后按空格键，然后按 Ctrl + Shift + V 组合键粘贴之前所复制的网址。

➢ **下载 Anaconda2-2.5.0-Linux-x86_64.sh**

```
wget https://repo.continuum.io/archive/Anaconda2-2.5.0-Linux-x86_64.sh
```

运行结果如图 9-2 所示。

```
hduser@master: ~
hduser@master:~$ wget https://repo.continuum.io/archive/Anaconda2-2.5.0-Linux
-x86_64.sh
--2016-08-10 13:21:17--  https://repo.continuum.io/archive/Anaconda2-2.5.0-Li
nux-x86_64.sh
正在解析主机 repo.continuum.io (repo.continuum.io)... 23.21.228.27, 54.243.35
.219, 54.163.242.177, ...
正在连接 repo.continuum.io (repo.continuum.io)|23.21.228.27|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 409842279 (391M) [application/octet-stream]
Saving to: 'Anaconda2-2.5.0-Linux-x86_64.sh'

100%[===================================>] 409,842,279 1.70MB/s   in 3m 50s

2016-08-10 13:25:08 (1.70 MB/s) - 'Anaconda2-2.5.0-Linux-x86_64.sh' saved [40
9842279/409842279]
```

图 9-2　下载 Anaconda2-2.5.0-Linux-x86-64.sh

步骤03 安装 Anaconda

在“终端”程序输入下列命令，执行 Anaconda2-2.5.0-Linux-x86_64.sh。

```
bash Anaconda2-2.5.0-Linux-x86_64.sh -b
```

运行后，屏幕显示结果如图 9-3 所示。

```
hduser@master: ~
hduser@master:~$ bash Anaconda2-2.5.0-Linux-x86_64.sh -b
PREFIX=/home/hduser/anaconda2
installing: _cache-0.0-py27_x0 ...
installing: python-2.7.11-0 ...
installing: abstract-rendering-0.5.1-np110py27_0 ...
installing: alabaster-0.7.7-py27_0 ...
installing: anaconda-client-1.2.2-py27_0 ...
installing: argcomplete-1.0.0-py27_1 ...
installing: astropy-1.1.1-np110py27_0 ...
installing: babel-2.2.0-py27_0 ...
installing: backports_abc-0.4-py27_0 ...
installing: beautifulsoup4-4.4.1-py27_0 ...
```

图 9-3　安装 Anaconda

以上指令加上“-b”是指 batch，即批次安装，会自动省略阅读 License 条款，自动安装到 / home/hduser/anaconda2 路径。

步骤04 编辑 ~/.bashrc 加入模块路径

可以在“终端”程序输入下列命令编辑~/.bashrc：

```
sudo gedit ~/.bashrc
```

运行后，显示结果如图 9-4 所示。

以上设置说明如下：

➢ 加入 Anaconda 路径

将 Anaconda 执行时的路径加入 PATH，可在不同路径执行 Anaconda。

```
export PATH=/home/hduser/anaconda2/bin:$PATH
export ANACONDA_PATH=/home/hduser/ anaconda2
```



图 9-4　编辑 ~/.bashrc 加入模块路径

➢ 加入 pyspark 设置

当我们执行 pyspark 时会使用下列 pyspark 设置。

```
export PYSPARK_DRIVER_PYTHON=$ANACONDA_PATH/bin/ipython
export PYSPARK_PYTHON=$ANACONDA_PATH/bin/python
```

步骤 05　使 ~/.bashrc 修改生效

我们可以重新注销再登录，或使用下列命令让用户环境变量的设置生效。

```
source ~/.bashrc
```

运行后结果如图 9-5 所示。

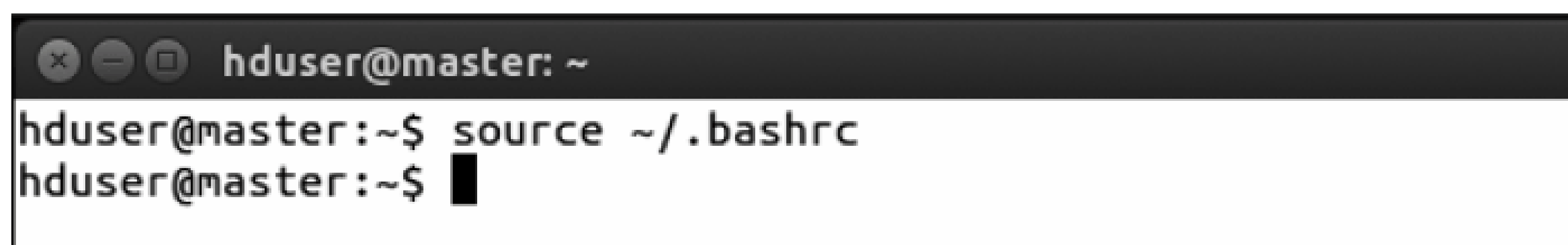


图 9-5　使 ~/.bashrc 修改生效

步骤 06　查看 Python 版本

可以在“终端”程序中输入下列命令：

```
python --version
```

运行后屏幕显示界面如图 9-6 所示，版本应该是 Anaconda 2.5.0。



图 9-6　查看 Python 版本

步骤 07　在 data1、data2、data3 安装 Anaconda

因为我们后续会在 cluster 模式运行 python spark 程序（该程序必须在 master、data1、

data2、data3 执行），所以必须在 data1、data2、data3 安装 Anaconda。

参照步骤 2~6，在 data1、data2、data3 安装 Anaconda。

➤ **data1 安装成功后查看版本**

```
hduser@data1: ~
hduser@data1:~$ python --version
Python 2.7.11 :: Anaconda 2.5.0 (64-bit)
```

➤ **data2 安装成功后查看版本**

```
hduser@data2: ~
hduser@data2:~$ python --version
Python 2.7.11 :: Anaconda 2.5.0 (64-bit)
```

➤ **data3 安装成功后查看版本**

```
hduser@data3: ~
hduser@data3:~$ python --version
Python 2.7.11 :: Anaconda 2.5.0 (64-bit)
```

9.2 在 IPython Notebook 使用 Spark

本节将介绍如何在 IPython Notebook 使用 Spark。

步骤 01 创建 ipynotebook 工作目录

可以在“终端”程序中输入下列命令，创建并切换 ipynotebook 工作目录。

```
mkdir -p ~/pythonwork/ipynotebook cd ~/pythonwork/ipynotebook
```

按 Enter 键后，运行结果如图 9-7 所示。

```
hduser@master: ~/pythonwork/ipynotebook
hduser@master:~$ mkdir -p ~/pythonwork/ipynotebook
hduser@master:~$ cd ~/pythonwork/ipynotebook
hduser@master:~/pythonwork/ipynotebook$
```

图 9-7 创建 ipynotebook 工作目录

步骤 02 在 IPython Notebook 界面中运行 pyspark

进入工作目录后，先在“终端”程序中输入下列命令，再在 IPython Notebook 界面运行 pyspark，默认是在本机运行 pyspark。

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888。打开 IPython Notebook 界面，如图 9-8 所示。



图 9-8　IPython Notebook 界面

步骤 03　新建一个 IPython Notebook

进入 IPython Notebook 界面，可以按照图 9-9 所示的步骤新建 Notebook。

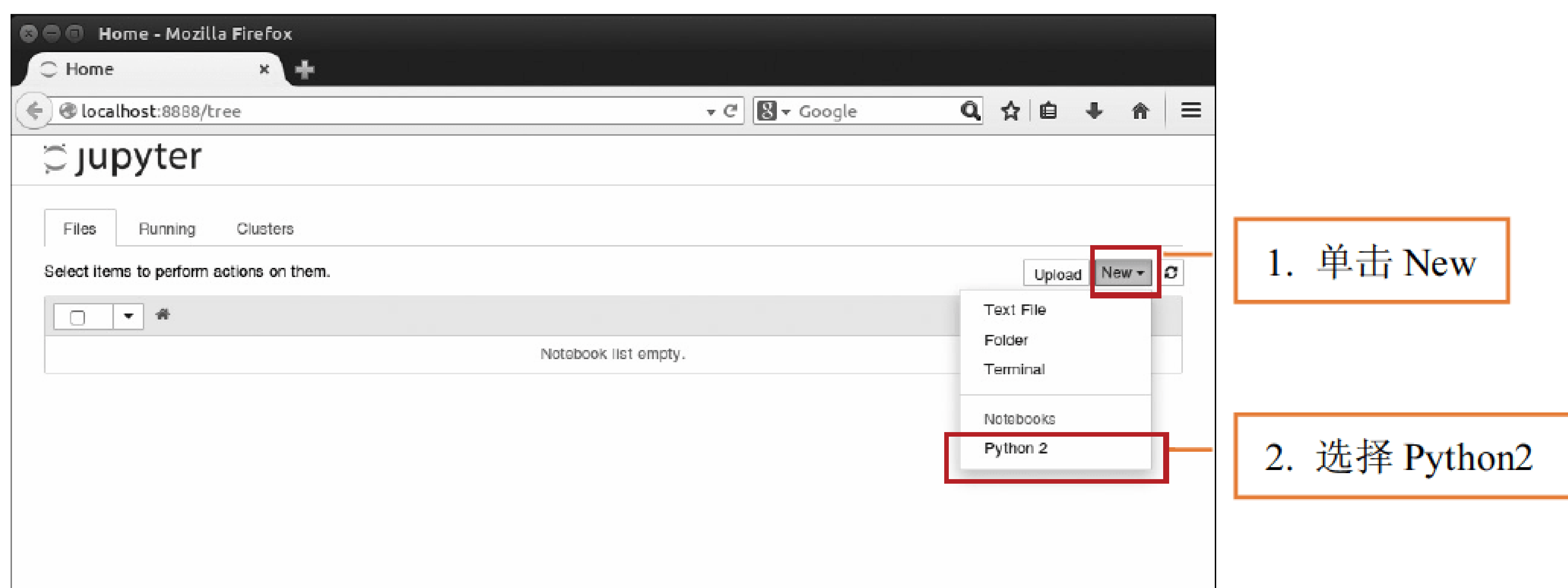


图 9-9　新建 Notebook

步骤 04　新建的 IPython Notebook

新建 IPython Notebook 后，会打开新的页面。默认的 Notebook 名称是 Untitled，我们可以用鼠标单击后修改 Notebook 名称，如图 9-10 所示。

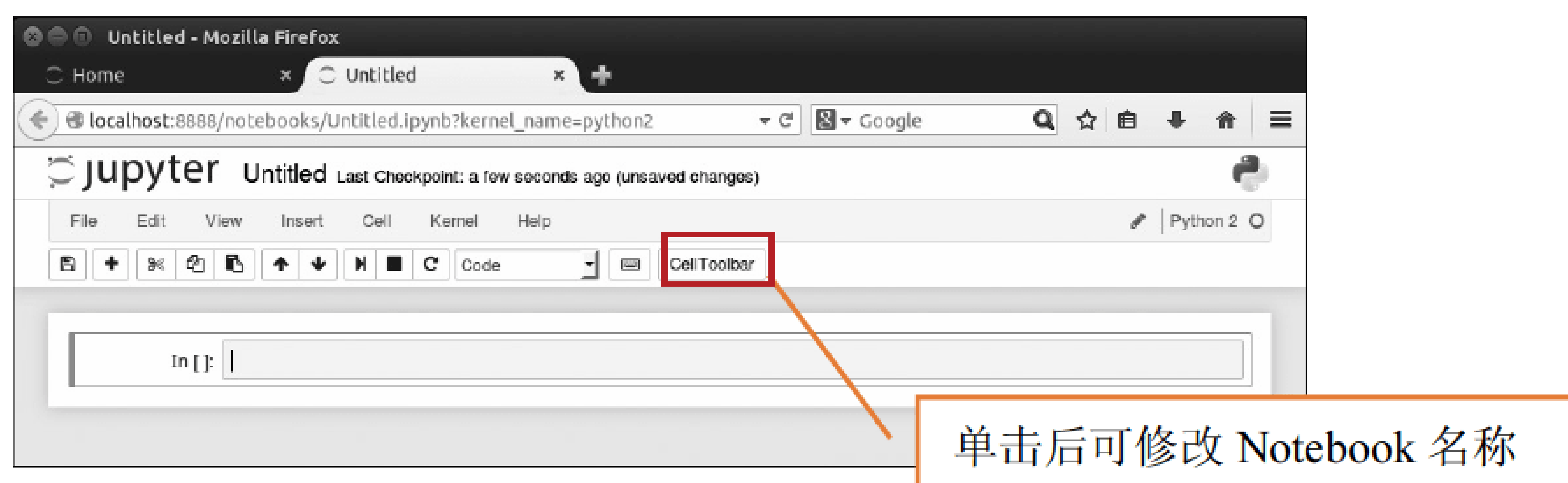


图 9-10　新建的 Notebook 界面

步骤 05　输入新的 Notebook 名字

用鼠标单击后，输入新的名字，再单击 OK 按钮，如图 9-11 所示。



图 9-11　输入新名字

步骤 06　在 IPython Notebook 运行程序代码

在 IPython Notebook 的 Cell（程序单元格）输入程序代码，然后按 Shift + Enter 或 Ctrl + Enter 组合键运行程序代码，这两种方式的差异如下。

- Shift + Enter：运行后，光标会移至下一个 Cell（程序单元格）。
- Ctrl + Enter：运行后，光标仍在当前的 Cell（程序单元格）。

运行结果如图 9-12 所示。

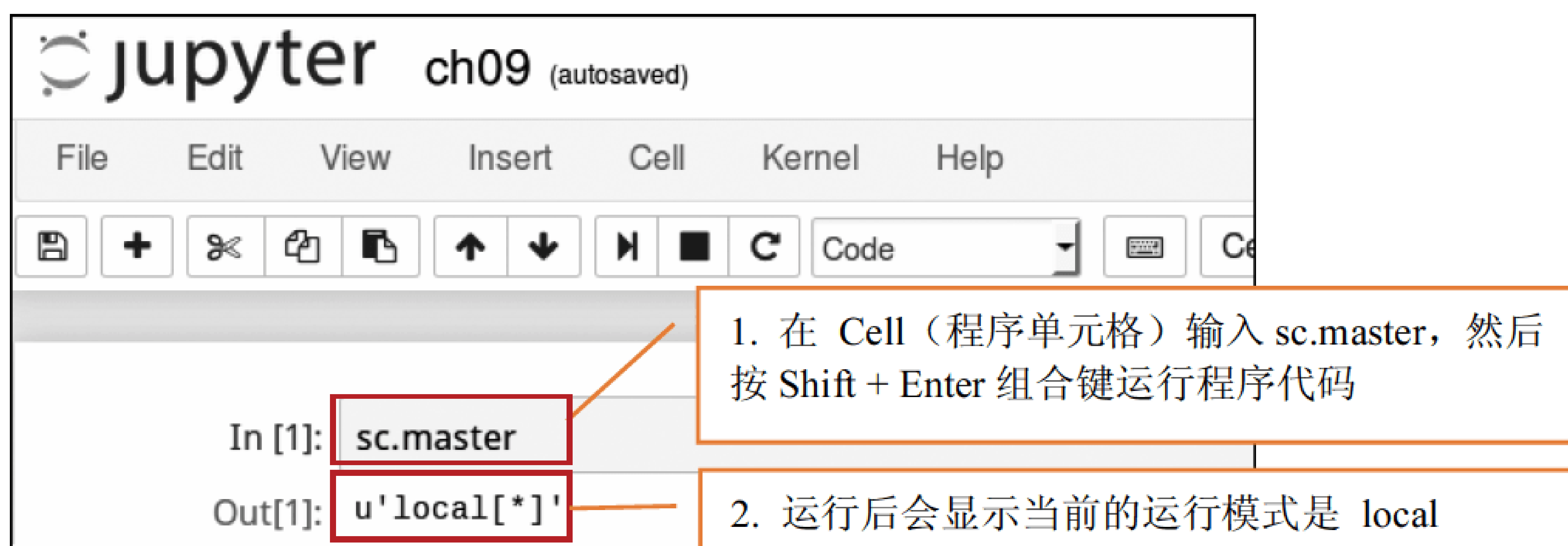


图 9-12　运行程序代码

步骤 07 IPython Notebook 运行后的屏幕显示结果

运行结果如图 9-13 所示。

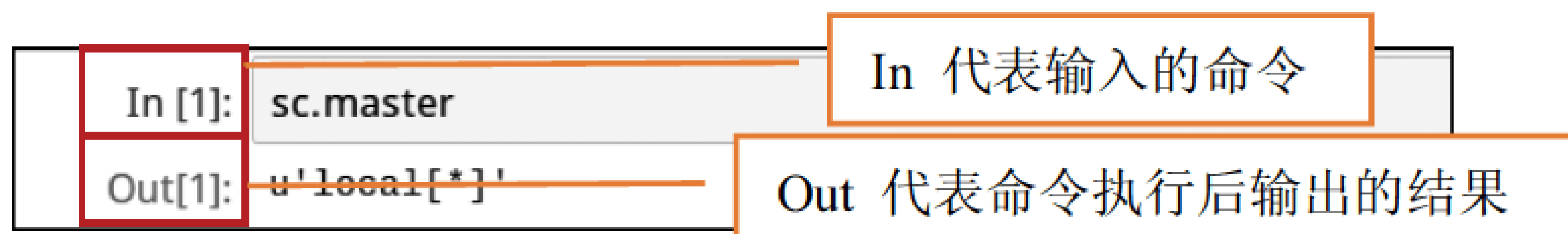


图 9-13 运行结果

步骤 08 读取本地文件程序代码

在 IPython Notebook 新的 Cell 输入如图 9-14 所示的程序代码来读取本地文件。

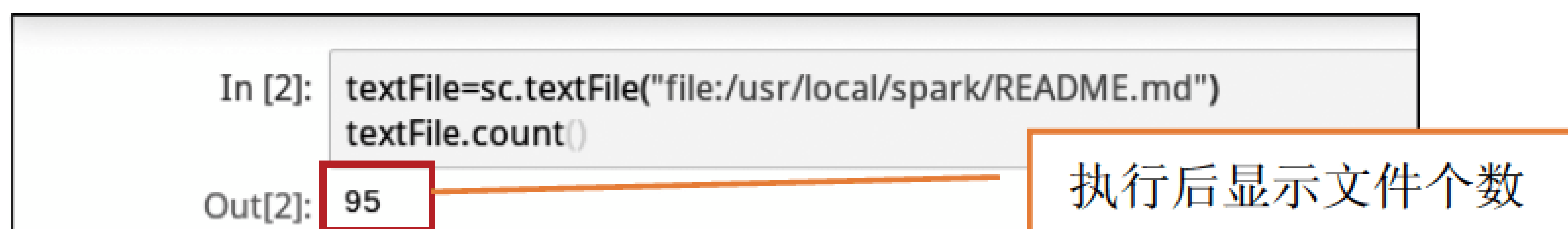


图 9-14 读取本地文件程序代码

步骤 09 输入读取 HDFS 文件程序代码

使用 sc.textFile 读取 HDFS 文件之前，要先启动 Hadoop 并将测试文本文件上传到 HDFS（请参考第 8.5 节的说明），在程序单元格输入下列命令，读取 HDFS 文件并查看文本文件共几项数据。运行后屏幕显示界面如图 9-15 所示。



图 9-15 读取 HDFS 文件程序代码

步骤 10 保存 Notebook

在退出 Notebook 之前，记得保存之前的内容，如图 9-16 所示。

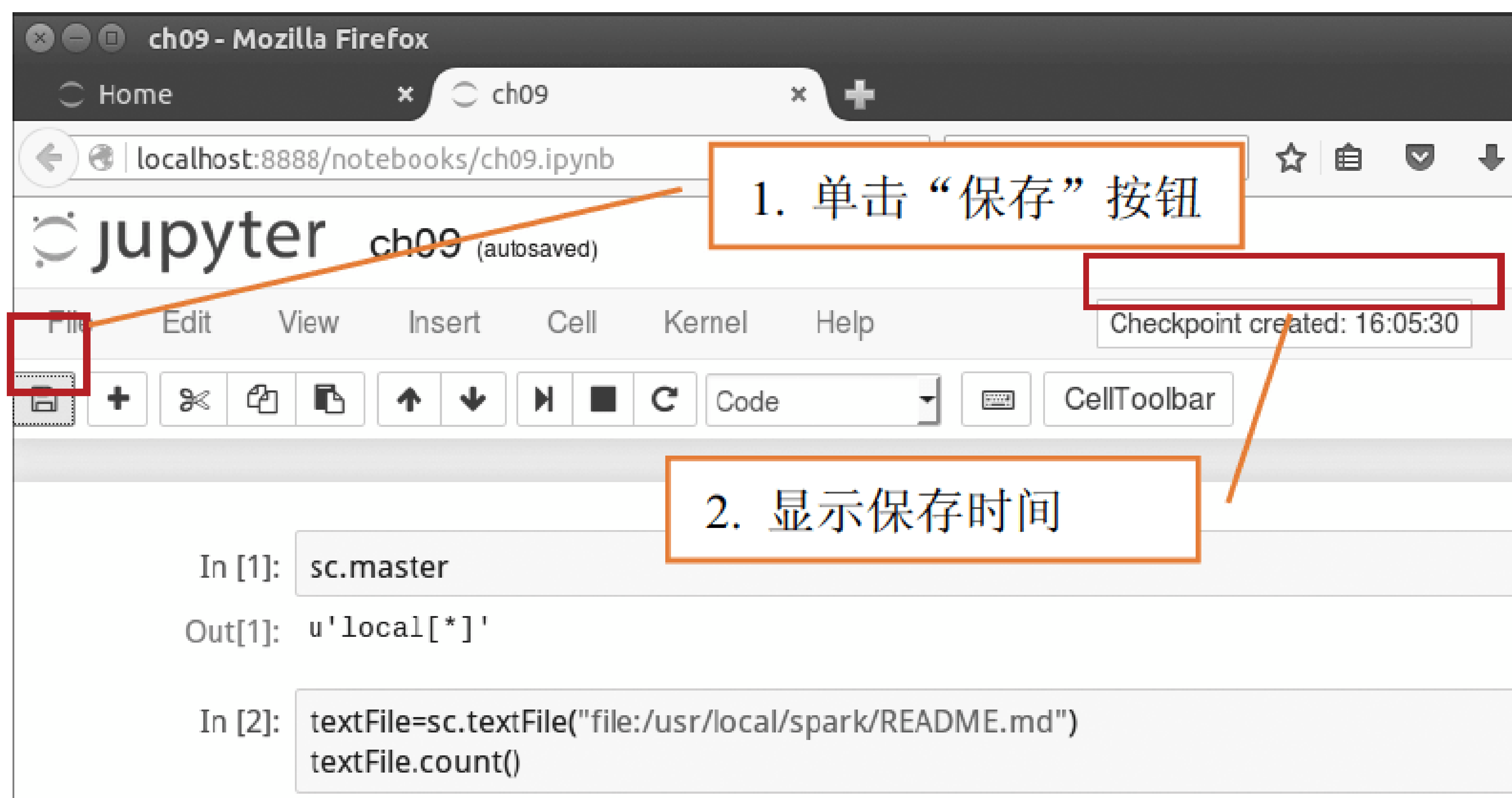


图 9-16 保存内容

步骤 11 关闭 Notebook 网页

保存完成后就可以关闭 Notebook 网页了，如图 9-17 所示。

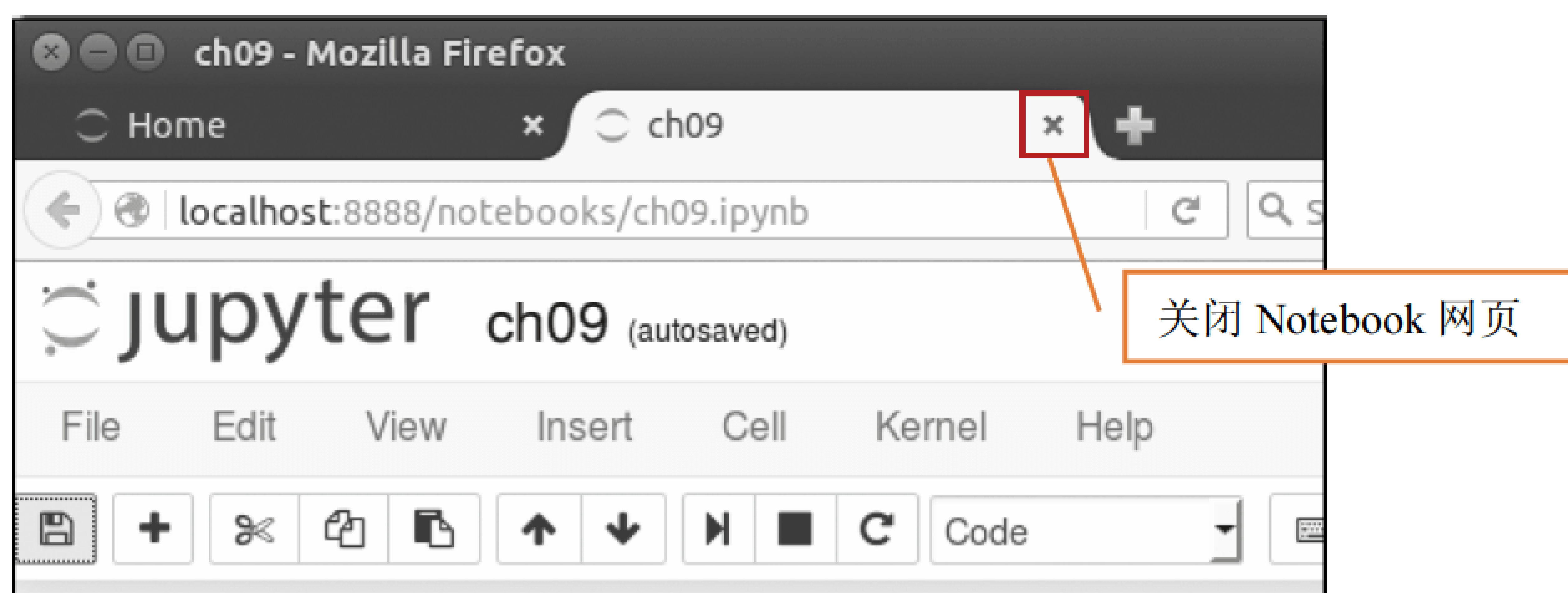


图 9-17 关闭 Notebook 网页

9.3 打开 IPython Notebook 笔记本

步骤 01 查看之前保存的 Notebook

回到 IPython NotebookHome 网页，我们可以看到之前保存的 ch09.ipynb，如图 9-18 所示。



图 9-18 查看之前保存的 Notebook

步骤 02 打开之前保存的 Notebook

打开之后，就可以看到之前保存的 Notebook 了，如图 9-19 所示。

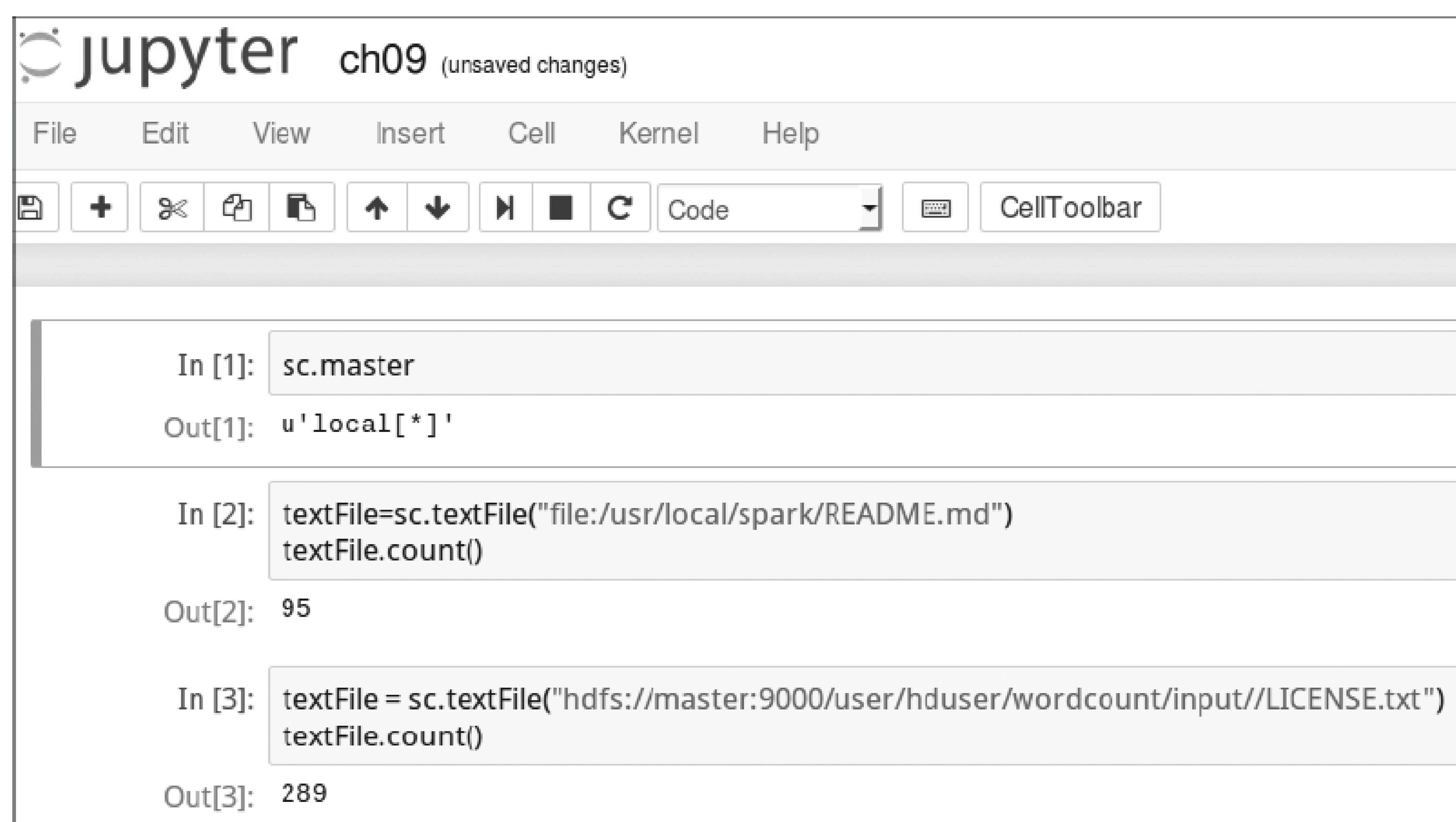


图 9-19　打开 Notebook

9.4 插入程序单元格

在 IPython Notebook 输入程序代码的文本框称为程序单元格（Cell），我们可以在单元格之前或之后插入新的单元格。

步骤 01 插入程序单元格的方式

有 3 种方式可用于插入单元格（参考图 9-20）。

- 利用 Insert → Insert Cell Above 菜单在当前单元格之前插入单元格。
- 利用 Insert → Insert Cell Below 菜单在当前单元格之后插入单元格。
- 单击“+”图标，在当前单元格之后插入单元格。

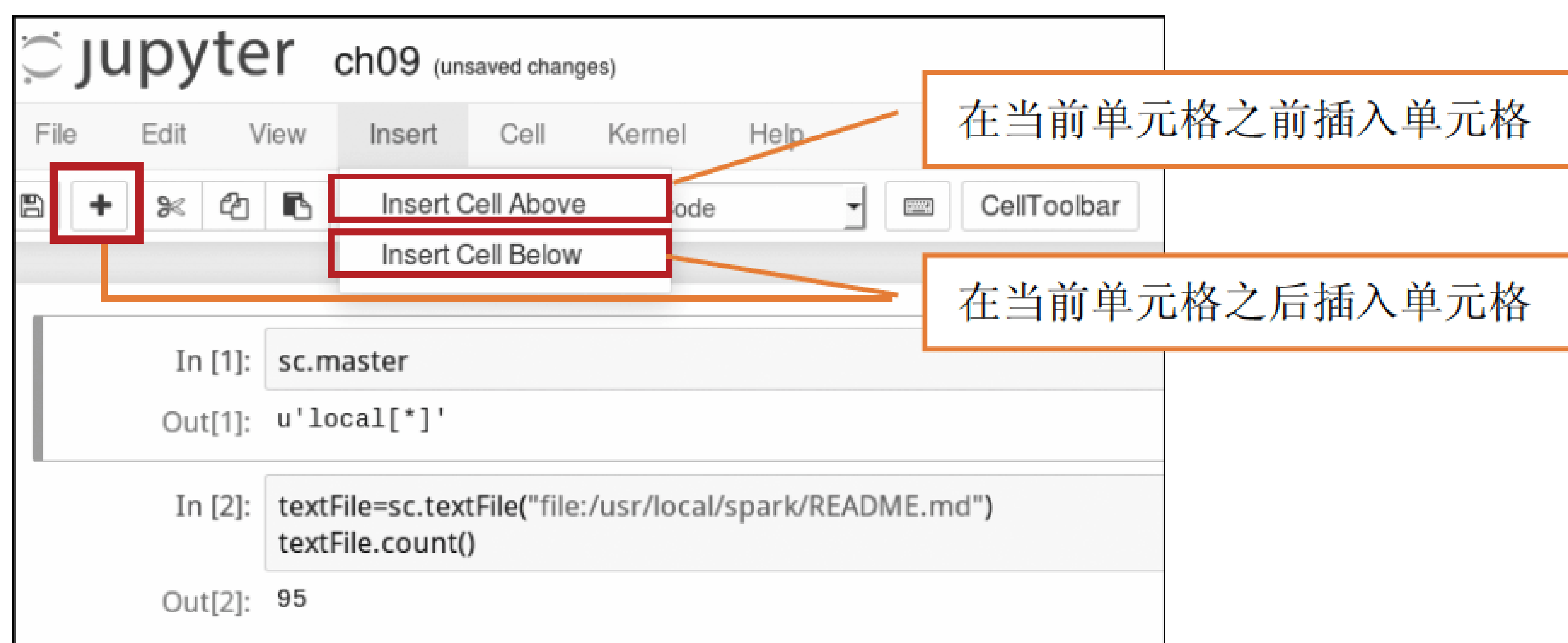


图 9-20　插入单元格的方法示意图

步骤02 新建程序单元格

我们选择 Insert → Insert Cell Above 菜单项，在当前单元格之前插入单元格，如图 9-21 所示。

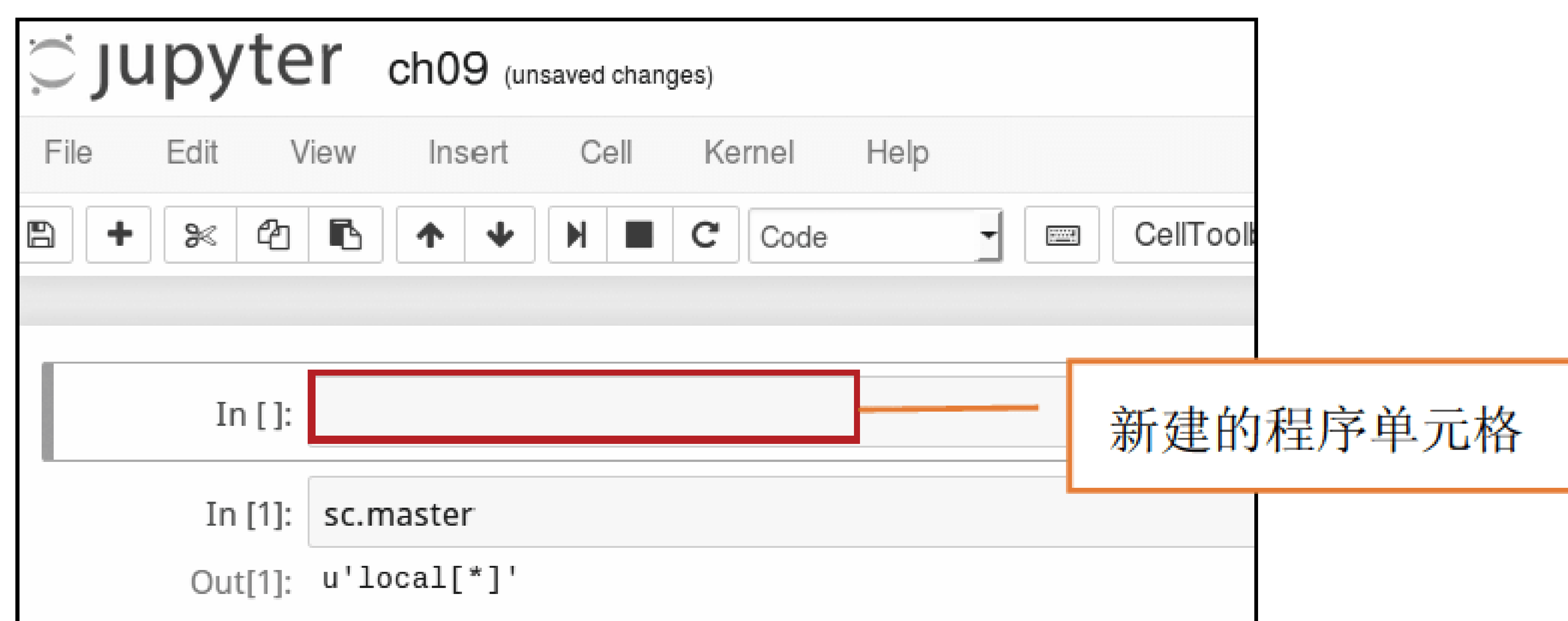


图 9-21 插入单元格

9.5 加入注释与设置程序代码说明标题

在 IPython Notebook 中，我们可以加入批注来说明程序的功能。

步骤01 输入注释

输入注释内容后要在前面加入#，如图 9-22 所示。



图 9-22 输入注释

步骤02 设置为标题

按照图 9-23 的步骤将之前输入的注释设置为标题。

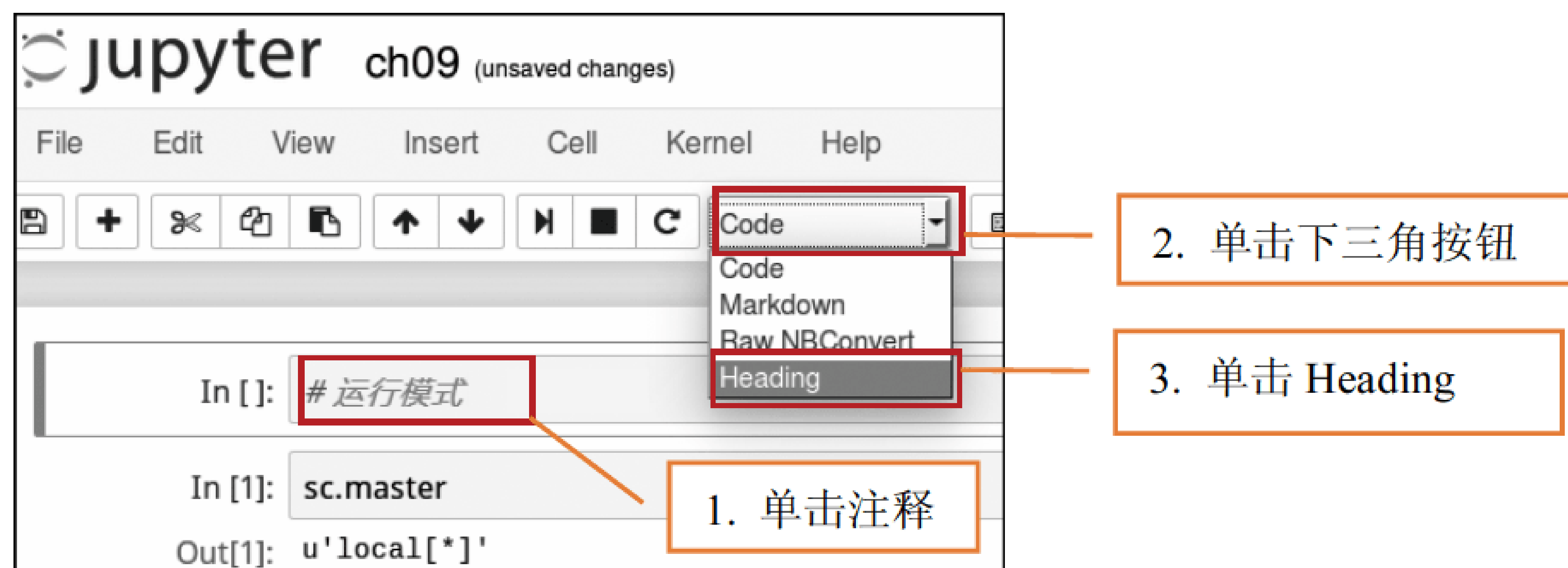


图 9-23 设置为标题

步骤 03 使用 MarkDown Heading 的说明（见图 9-24）

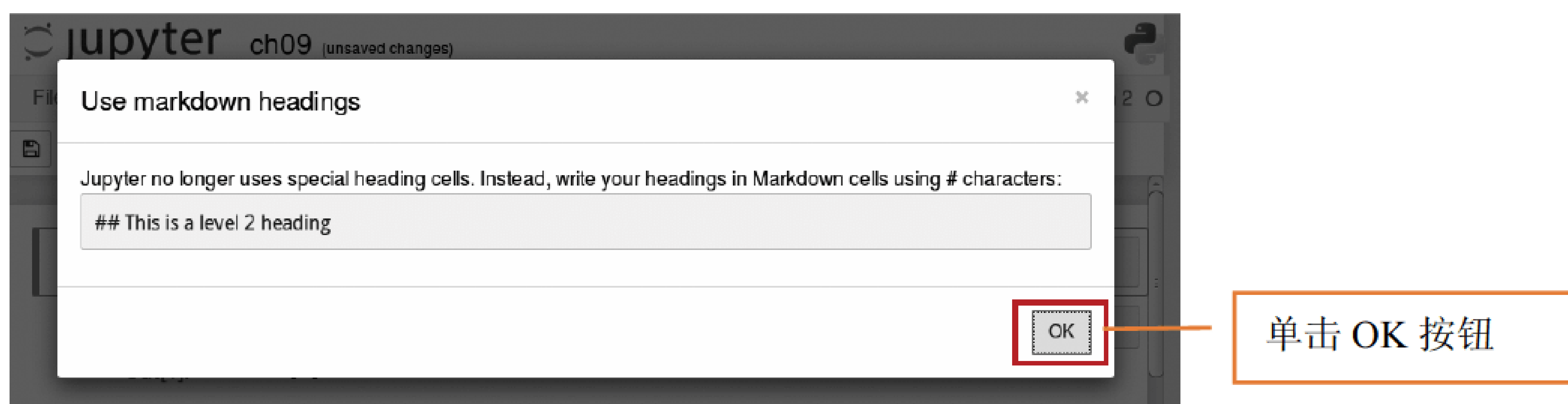


图 9-24　使用 MarkDown Heading 的说明

步骤 04 运行程序代码

设置为 Heading 标题后，我们可以看到字体变大了，再按 Shift + Enter 组合键即可运行，如图 9-25 所示。



图 9-25　运行程序代码

步骤 05 设置完成后的标题（见图 9-26）

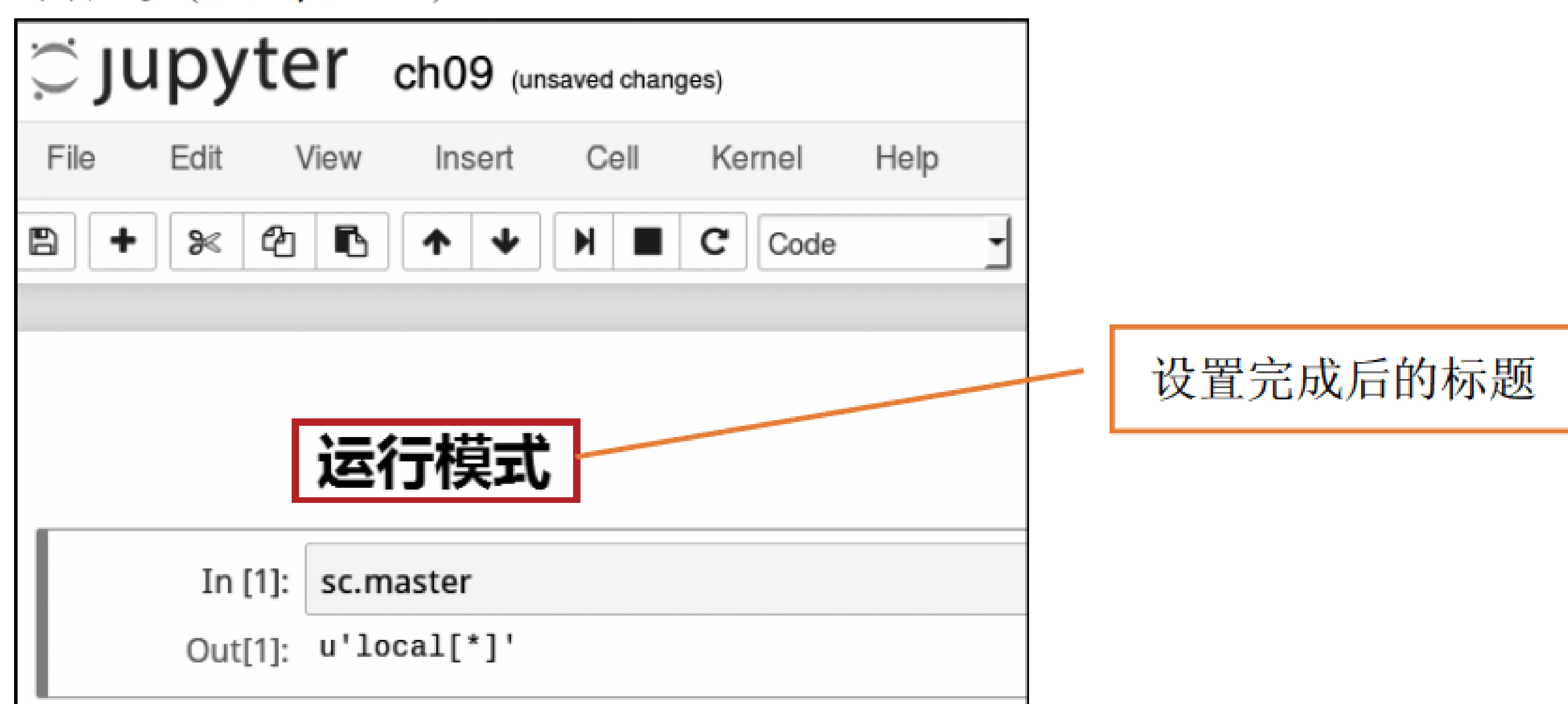


图 9-26　设置完成后的标题

步骤 06 按照之前的步骤输入标题

按照之前的步骤 1~5 输入标题，完成后如图 9-27 所示。这样我们可以看到每一段程序代

码都有标题说明，让整个 IPython Notebook 更清楚明了。

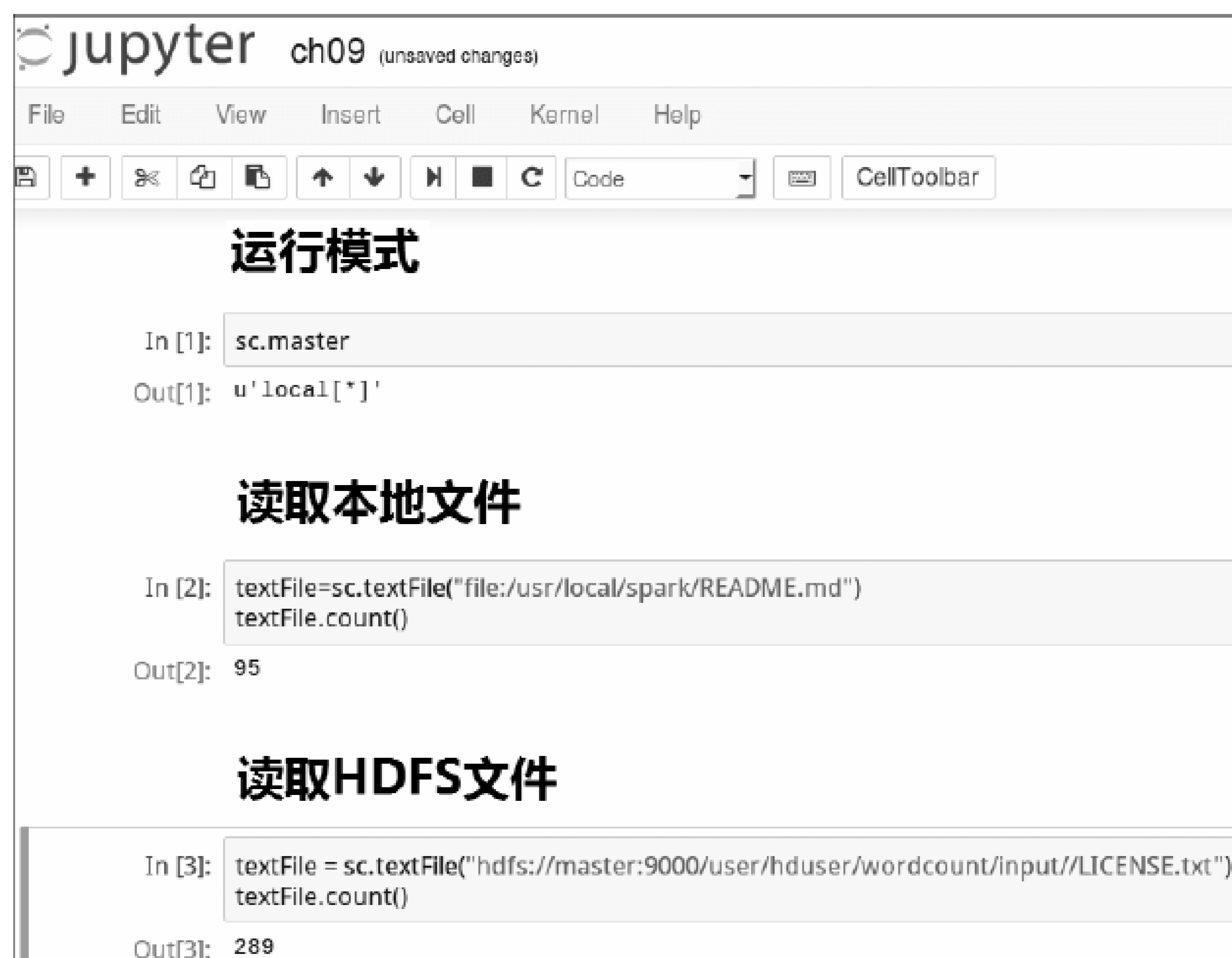


图 9-27　继续输入标题

9.6 关闭 IPython Notebook

步骤01 关闭浏览器

如果打开了多个 Notebook，建议逐个关闭，而不要直接关掉浏览器。因为关闭 Notebook 时，如果内容已经修改而未保存，系统会提醒是否确定要退出，如图 9-28 所示。如果直接关闭了浏览器，系统就不会询问了。

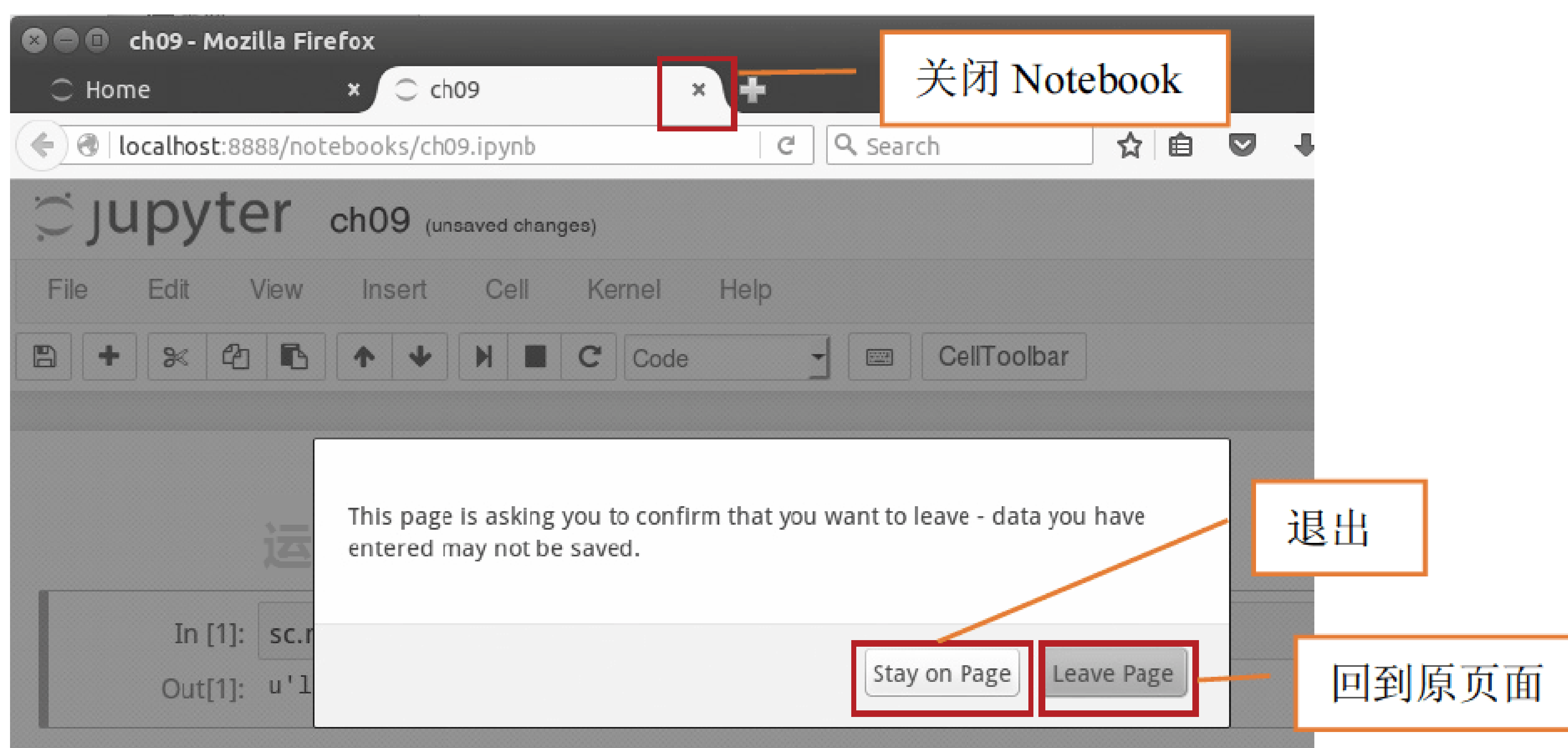


图 9-28　关闭浏览器

步骤02 关闭IPython Notebook

关闭浏览器后回到“终端”程序界面，可以按 Ctrl + C 组合键关闭 IPython Notebook 程序，

过程如图 9-29 所示。

```
hduser@master: ~/pythonwork/ipynotebook
hduser@master:~/pythonwork/ipynotebook$ PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DR
IVER_PYTHON_OPTS="notebook" pyspark
[I 16:34:12.208 NotebookApp] Serving notebooks from local directory: /home/hduse
r/pythonwork/ipynotebook
[I 16:34:12.208 NotebookApp] 0 active kernels
[I 16:34:12.208 NotebookApp] The Jupyter Notebook is running at: http://localhos
t:8888/
[I 16:34:12.209 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
^C[I 16:34:55.133 NotebookApp] interrupted
Serving notebooks from local directory: /home/hduser/pythonwork/ipynotebook
0 active kernels
The Jupyter Notebook is running at: http://localhost:8888/
Shutdown this notebook server (y/[n])? No answer for 5s: resuming operation...
```

按 Ctrl + C 组合键，输入 y

图 9-29　关闭 Notebook

9.7 使用 IPython Notebook 在 Hadoop YARN-client 模式运行

接下来，我们将使用 IPython Notebook 在 Hadoop YARN-client 模式执行 Spark 程序。

步骤 01 启动 IPython Notebook 运行在 Hadoop YARN-client 模式

在“终端”程序中输入下列命令：

- **启动 Hadoop cluster**

参考第 8.6 节的内容，先启动 master、data1、data2、data3，然后执行 start-all.sh。

```
start-all.sh
```

- **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

- **IPython Notebook 运行在 Hadoop YARN-client 模式**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook"
HADOOP_CONF_IR=/usr/local/hadoop/etc/hadoop MASTER=yarn-client pyspark
```

按 Enter 键后会启动浏览器，这时显示 IPython Notebook 界面，如图 9-30 所示。

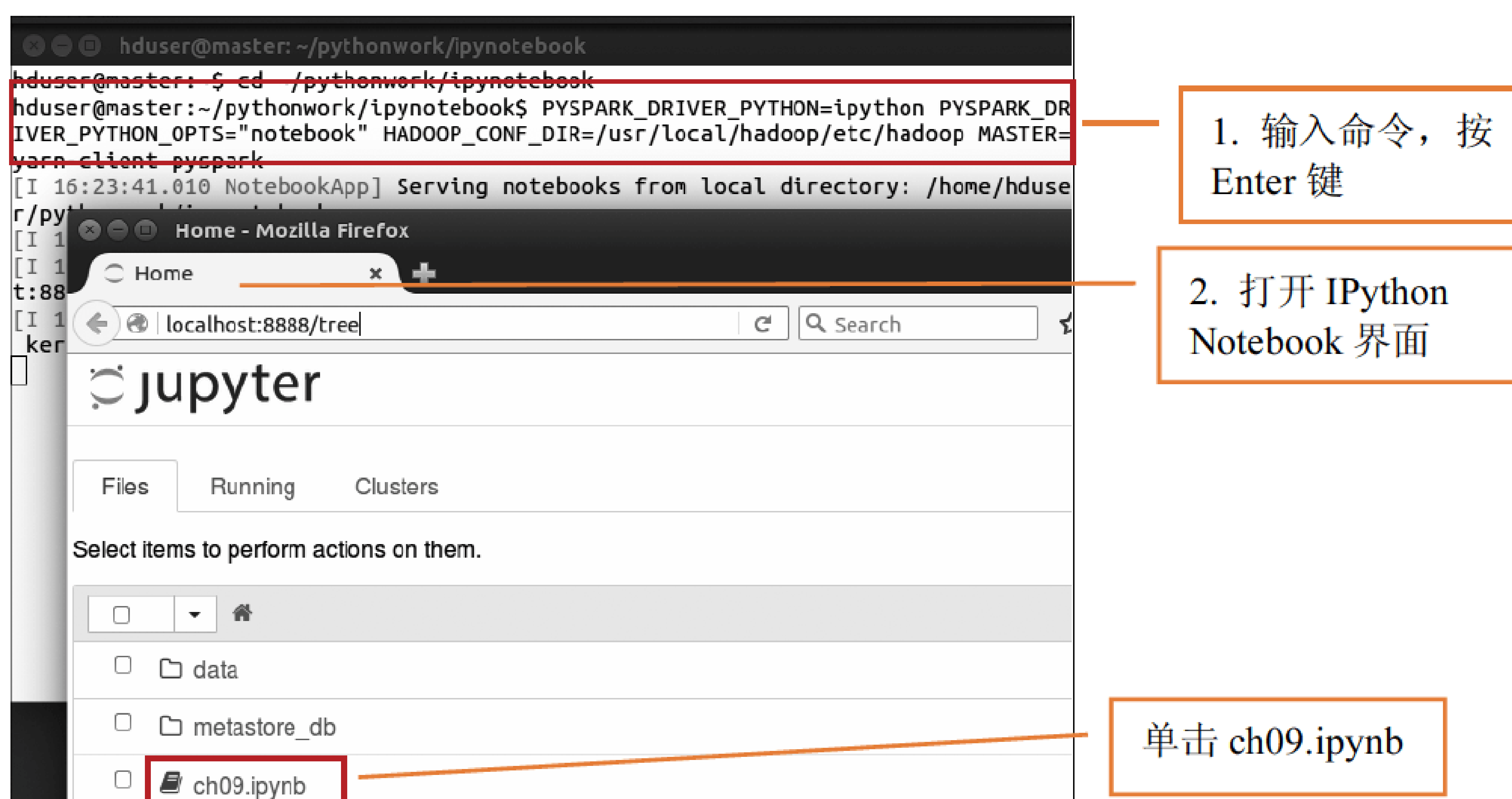


图 9-30　启动 IPython Notebook 运行在 Hadoop YARN-client 模式

步骤 02　全部重新执行 Notebook 程序代码

打开 ch09 Notebook 之后，可以依次选择 Cell → Run All 菜单选项，全部重新运行 Notebook 程序代码，如图 9-31 所示。

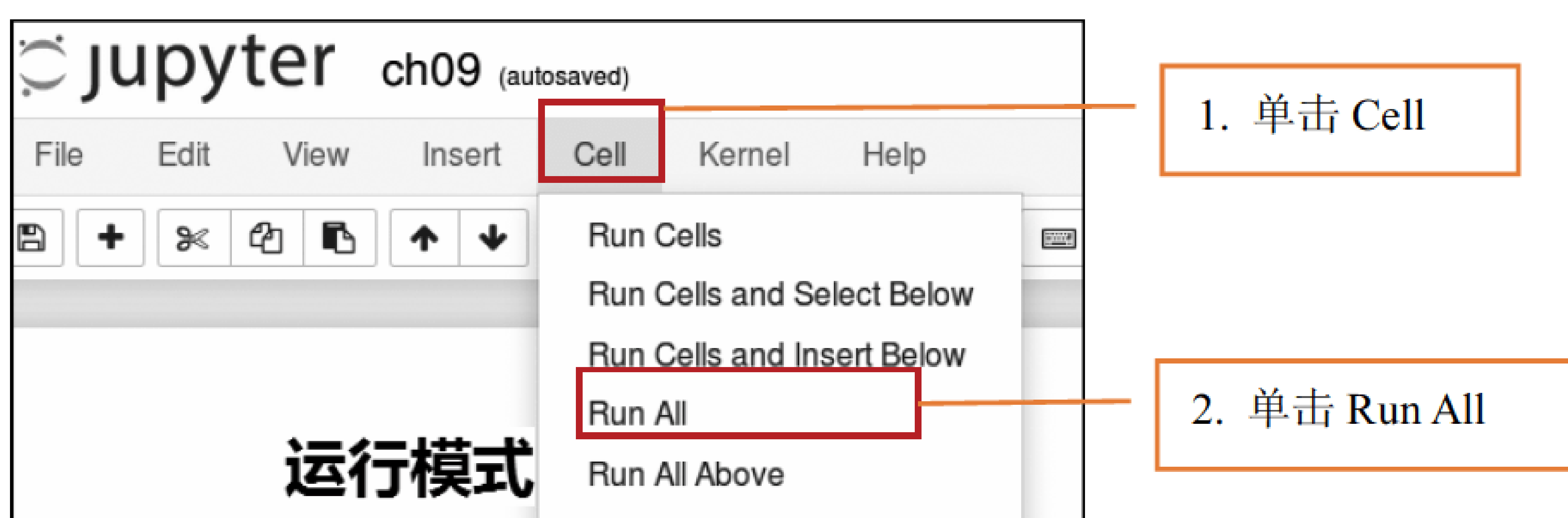


图 9-31　全部重新执行 Notebook 程序代码

步骤 03　程序代码运行中（见图 9-32）

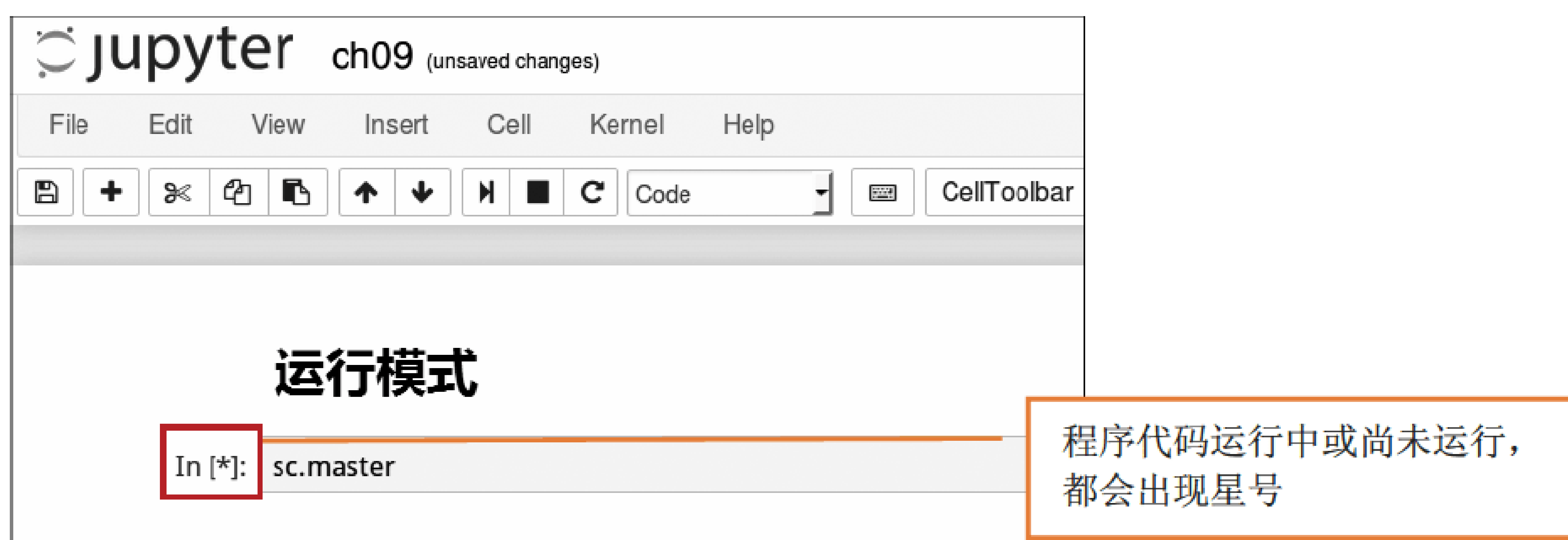


图 9-32　程序代码运行中

步骤 04　程序代码运行完成（见图 9-33）



图 9-33　程序代码运行完成

步骤 05　在 Hadoop Web 界面查看 pyspark App

现在我们已经在 Hadoop YARN 运行了 pyspark，所以可以在 Hadoop Web 界面看到这个应用程序（Application）。请按照图 9-34 所示的步骤打开 Hadoop Web 界面。

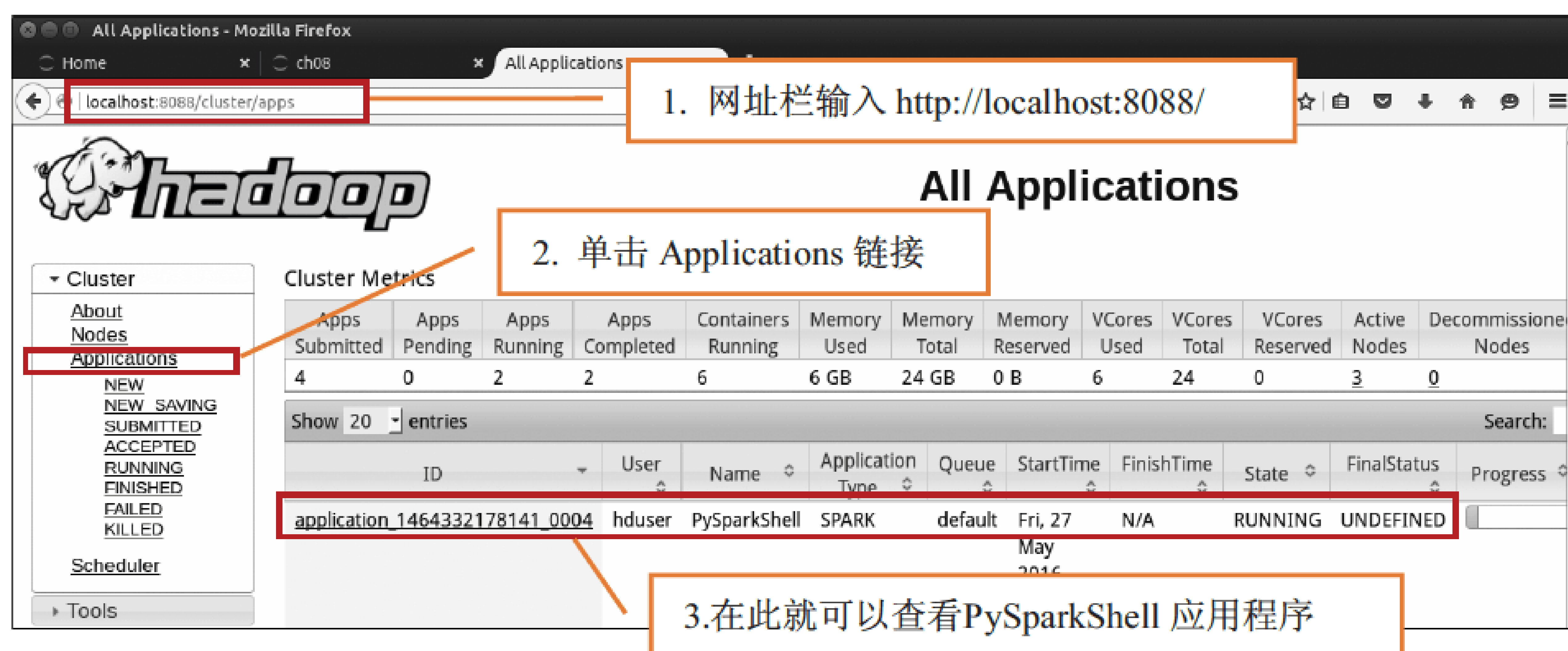


图 9-34　在 Hadoop Web 界面查看 pyspark App

步骤 06　关闭 IPython Notebook

关闭浏览器后回到“终端”程序的界面，按 Ctrl + C 组合键来关闭 IPython Notebook 程序，如图 9-35 所示。

```
hduser@master: ~/pythonwork/ipynotebook
[I 16:23:41.011 NotebookApp] The Jupyter Notebook is running at: http://localhos
t:8888/
[I 16:23:41.011 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[I 16:25:44.924 NotebookApp] Kernel started: 4a1ca0b2-380e-4d05-804d-9c146ae97bf
1
16/07/18 16:25:48 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
16/07/18 16:25:52 WARN YarnClientSchedulerBackend: NOTE: SPARK_WORKER_MEMORY is
deprecated. Use SPARK_EXECUTOR_MEMORY or --executor-memory through spark-submit
instead.
16/07/18 16:25:52 WARN YarnClientSchedulerBackend: NOTE: SPARK_WORKER_CORES is d
eprecated. Use SPARK_EXECUTOR_CORES or --executor-cores through spark-submit ins
tead.
[W 16:25:56.007 NotebookApp] Timeout waiting for kernel_info reply from 4a1ca0b2
-380e-4d05-804d-9c146ae97bf1
[I 16:27:44.912 NotebookApp] Saving file at /ch09.ipynb
[I 16:28:15.755 NotebookApp] Saving file at /ch09.ipynb
^C[I 16:28:29.285 NotebookApp] interrupted
Serving notebooks from local directory: /home/hduser/pythonwork/ipynotebook
1 active kernels
The Jupyter Notebook is running at: http://localhost:8888/
Shutdown this notebook server (y/[n])? No answer for 5s: resuming operation...
```

按 Ctrl + C 组合键，然后输入 y

图 9-35　关闭 IPython Notebook

9.8 使用 IPython Notebook 在 Spark Stand Alone 模式运行

步骤 01 启动 Spark Stand Alone cluster

在“终端”程序中输入下列命令，启动 Spark Stand Alone cluster。

```
/usr/local/spark/sbin/start-all.sh
```

运行后屏幕显示界面如图 9-36 所示。

```
hduser@master: ~
hduser@master:~$ /usr/local/spark/sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/sbin
/../logs/spark-hduser-org.apache.spark.deploy.master.Master-1-master.out
data1: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data1.out
data2: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data2.out
data3: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/sbin/../logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data3.out
```

图 9-36　启动 Spark Stand Alone cluster

步骤 02 启动 IPython Notebook 运行在 Spark Stand Alone 模式

在“终端”程序中输入下列命令：

- **切换到 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **运行 IPython Notebook 以使用 Spark**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" MASTER=
spark://master:7077 pyspark --num-executors 1 --total-executor-cores 2
--executor-memory 512m
```

按 Enter 键后就会启动浏览器，如图 9-37 所示为 IPython Notebook 界面。

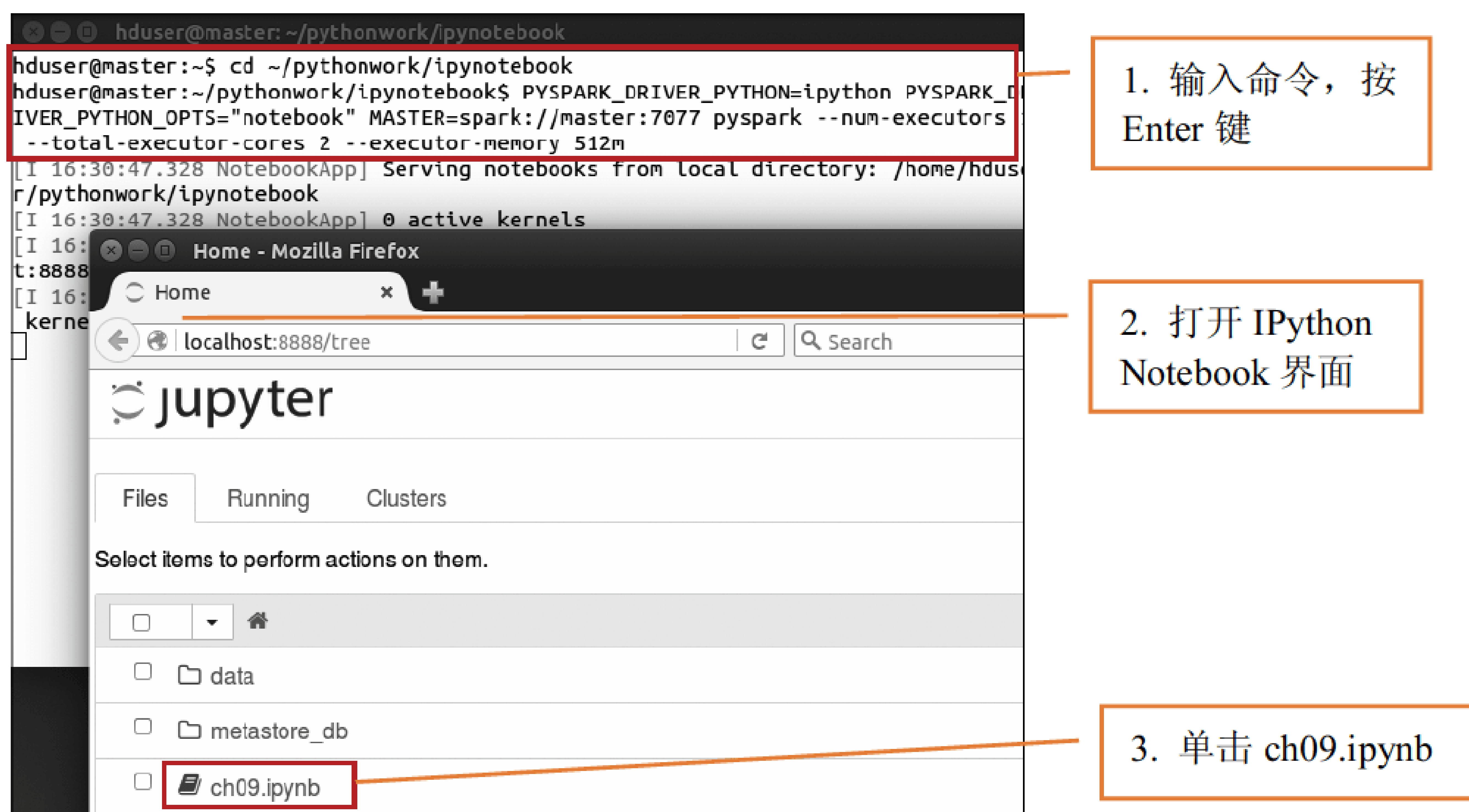


图 9-37　IPython Notebook 界面

步骤 03　全部重新执行 Notebook 程序代码

打开 ch09 Notebook 之后，可以依次选择 Cell → Run All 菜单选项，全部重新执行 Notebook 程序代码，如图 9-38 所示。

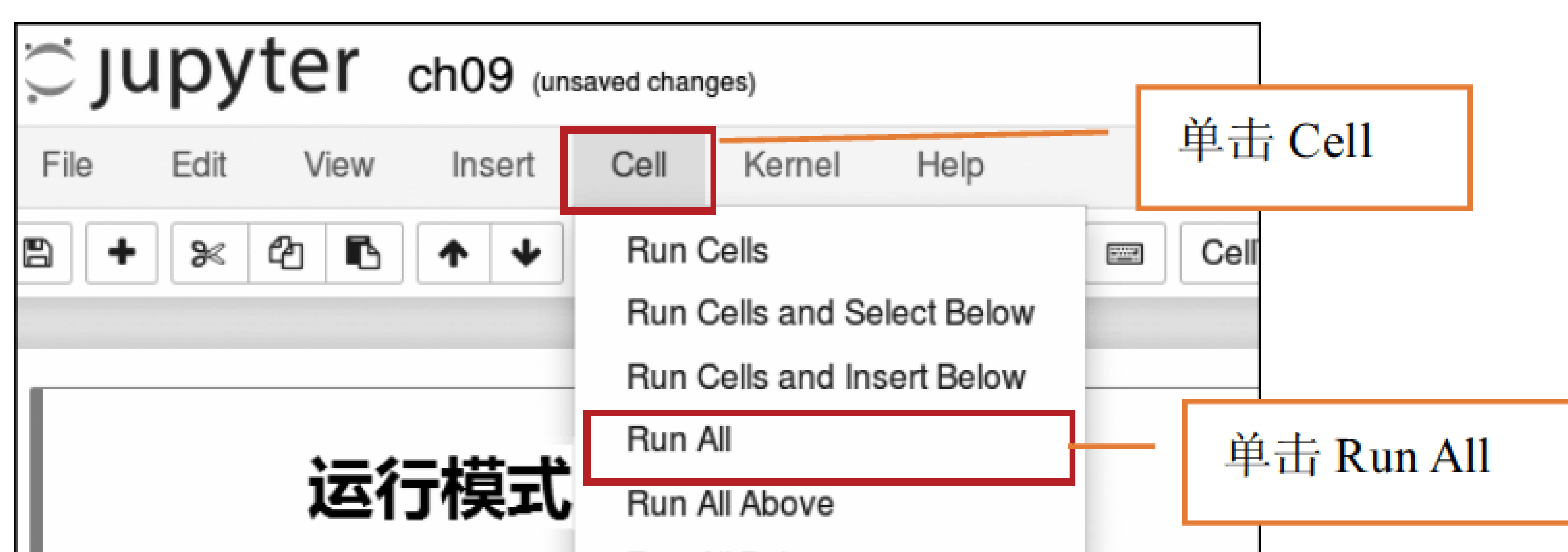


图 9-38　全部重新执行 Notebook 程序代码

步骤 04　程序代码运行完成（见图 9-39）

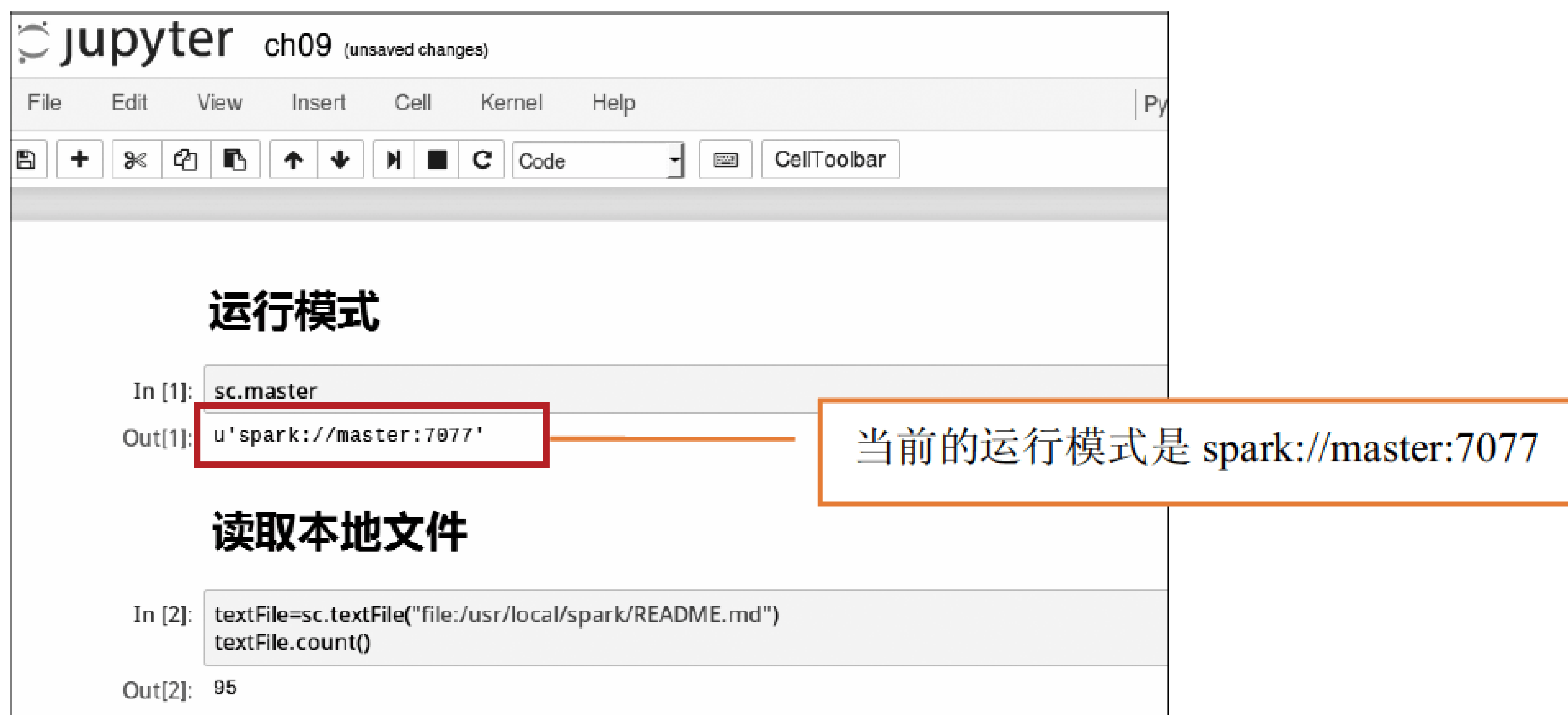


图 9-39　程序代码运行完成

步骤 05　查看 Spark Standalone Web UI 界面

按照如图 9-40 所示的步骤查看 Spark Standalone Web UI 界面，就可以看到当前正运行的应用程序 PySparkShell。

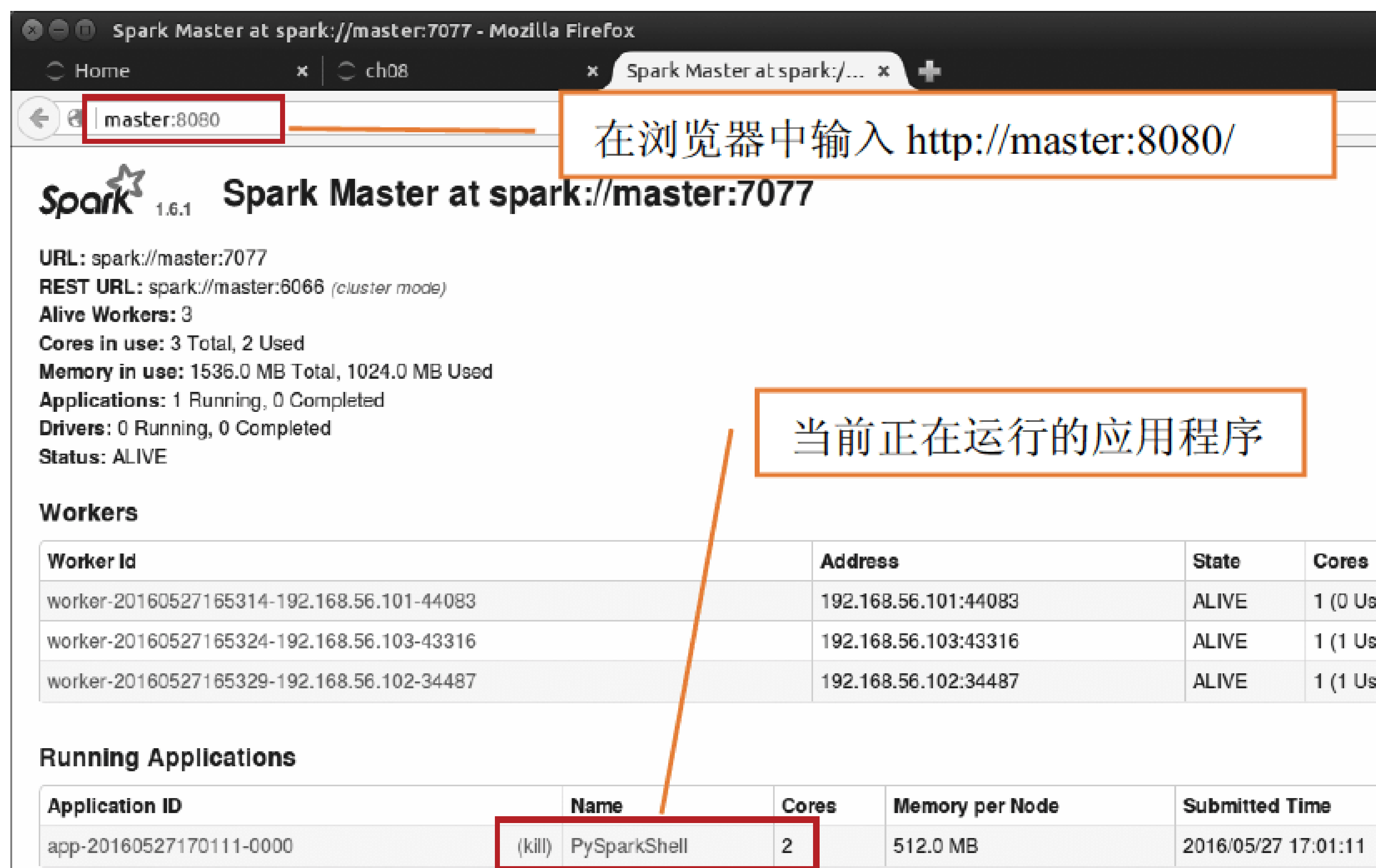


图 9-40　查看 Spark Standalone Web UI 界面

9.9　整理在不同的模式运行 IPython Notebook 的命令

后续章节为了让大家更容易理解 Spark，我们使用 IPython Notebook 做示范，在此整理

不同的模式运行 IPython Notebook 命令。读者可以选择自己要运行的模式。

9.9.1　在 Local 启动 IPython Notebook

在“终端”程序中输入下列命令，进入 IPython Notebook 交互式界面。

➢ **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **运行 IPython Notebook 以使用 Spark**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
--master local[*]
```

9.9.2　在 Hadoop YARN-client 模式启动 IPython Notebook

➢ **启动 hadoop cluster**

先在 VirtualBox 启动 master、data1、data2、data3 虚拟机，然后在 master“终端”程序中执行 start-all.sh。

```
start-all.sh
```

➢ **在 Hadoop YARN-client 模式运行 IPython Notebook**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" HADOOP_
CONF_DIR=/usr/local/hadoop/etc/hadoop pyspark --master yarn --deploy-mode client
```

9.9.3　在 Spark Stand Alone 模式启动 IPython Notebook

➢ **启动 hadoop cluster**

```
start-all.sh
```

➢ **启动 Spark Stand Alone cluster**

```
/usr/local/spark/sbin/start-all.sh
```

➢ **在 Spark Stand Alone 模式启动 IPython Notebook**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" MASTER=
spark://master:7077 pyspark --num-executors 1 --total-executor-cores 3
--executor-memory 512m
```

9.10 结论

本章我们介绍了 Anaconda 的安装，并且示范使用 IPython Notebook 在不同模式下运行、读取本地与 HDFS 文件，并且执行的命令与结果都可以存储在 IPython Notebook 文件中。接下来，我们将使用 IPython Notebook 介绍 Spark 基本功能 RDD。

第 10 章 Python Spark RDD

Spark 的核心是 RDD（Resilient Distributed Dataset），即弹性分布式数据集，是由 AMPLab 实验室提出的概念，属于一种分布式的内存系统数据集应用。Spark 的主要优势来自 RDD 本身的特性，RDD 能与其他系统兼容，可以导入外部存储系统的数据集，例如 HDFS、HBase 或其他 Hadoop 数据源。

10.1 RDD 的特性

➢ RDD 的 3 种基本运算

在 RDD 之上，可以施加 3 种类型的运算，如表 10-1 所示。

表 10-1　RDD 的基本运算

RDD 运算类型	说明
“转换”运算 Transformation	• RDD 执行“转换”运算的结果，会产生另外一个 RDD • RDD 具有 lazy 特性，所以“转换”运算并不会立刻实际执行，等到执行“动作”运算才会实际执行
“动作”运算 Action	• RDD 执行“动作”运算后不会产生另外一个 RDD，而是会产生数值、数组或写入文件系统 • RDD 执行“动作”运算时会立刻实际执行，并且连同之前的“转换”运算一起执行
“持久化” Persistence	对于那些会重复使用的 RDD，可以将 RDD“持久化”在内存中作为后续使用，以提高执行性能

RDD 通过“转换”运算可以得出新的 RDD，但 Spark 会延迟这个“转换”动作的发生时间点。它并不会马上执行，而是等到执行了 Action 之后才会基于所有的 RDD 关系来执行转换。示意图如图 10-1 所示。

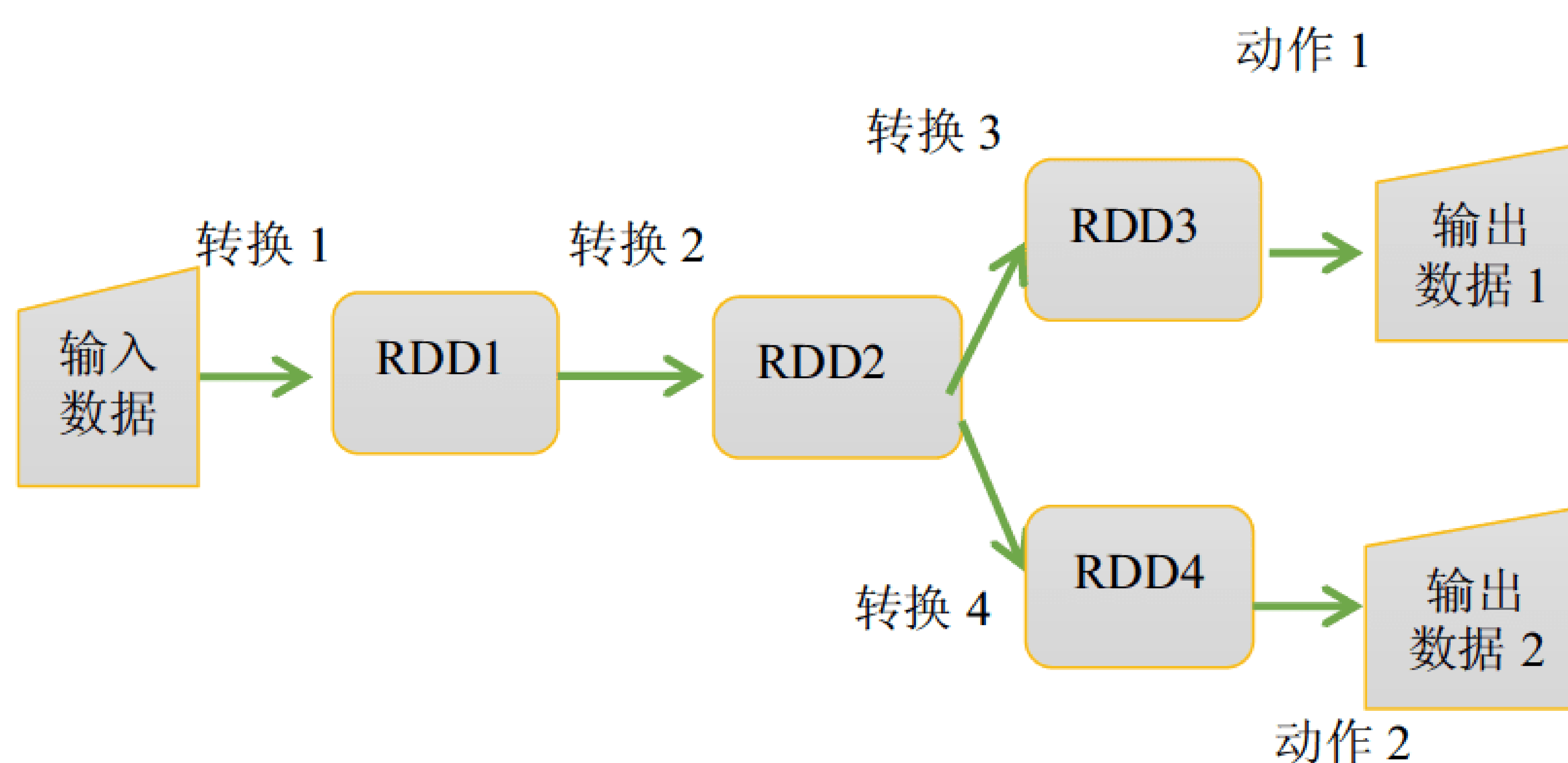


图 10-1　RDD 转换示意图

图 10-1 的说明如下：

- 输入数据，执行“转换 1”运算产生 RDD1，此时不会实际执行，只记录操作命令。
- RDD1 执行“转换 2”运算产生 RDD2，此时不会实际执行，只记录操作命令。

- RDD2 执行“转换 3”运算产生 RDD3，此时不会实际执行，只记录操作命令。
- RDD2 执行“转换 4”运算产生 RDD4，此时不会实际执行，只记录操作命令。
- RDD3 执行“动作 1”运算，此时会实际执行：“转换 1”+“转换 2”+“转换 3”+“动作 1”，产生输出数据 1。
- RDD4 执行“动作 2”运算，此时会实际执行：“转换 1”+“转换 2”+“转换 4”+“动作 2”，产生输出数据 2（如果之前已经先执行了“动作 1”运算，那么“转换 1”+“转换 2”已经实际执行完成，在此只会实际执行“转换 4”+“动作 2”，以节省运行时间）。

➢ **Lineage 机制具备容错的特性**

RDD 本身具有 Lineage 机制。它会记录每个 RDD 与其父代 RDD 之间的关联，还会记录通过什么操作才由父代 RDD 得到该 RDD 的信息。

RDD 本身的 immutable（不可变）特性，再加上 Lineage 机制，使得 Spark 具备容错的特性。如果某节点的机器出现了故障，那么存储于这个节点上的 RDD 损毁后就会重新执行一连串的“转换”命令，产生新的输出数据，以避免因为特定节点的故障而造成整个系统无法运行的问题。

➢ **RDD 命令整理**

本章的 RDD 命令已整理在本书的博客文章中。当练习安装时，可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可节省打字的时间，也不用担心打错字（无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书博客的网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

10.2 开启 IPython Notebook

为了让大家更容易理解 RDD 运算，我们使用 IPython Notebook 做示范。IPython Notebook 具有互动性，可以从中看到命令运行后的结果。读者可以参考第 9.9 节在不同模式启动 IPython Notebook 的情况。本节我们只示范在本地模式运行。

步骤 01 在 local 模式运行 IPython Notebook 来使用 Spark

在“终端”程序中输入下列命令：

➢ **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **运行 IPython Notebook 来使用 Spark**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888。图 10-2 所示的界面是 iPython NoteBook 界面。

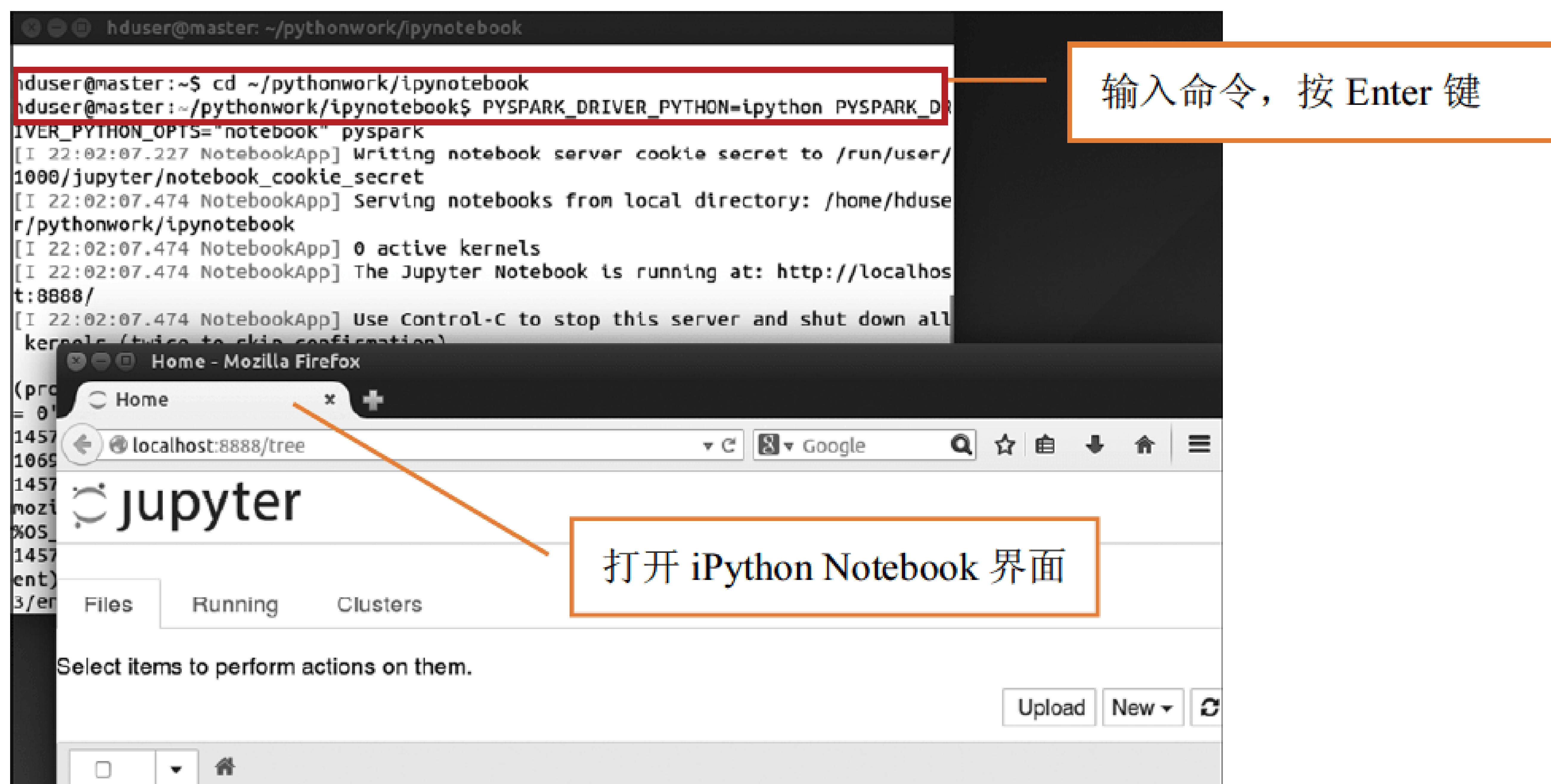


图 10-2 打开 IPython Notebook 界面

步骤 02 新建一个 Python Notebook（见图 10-3）

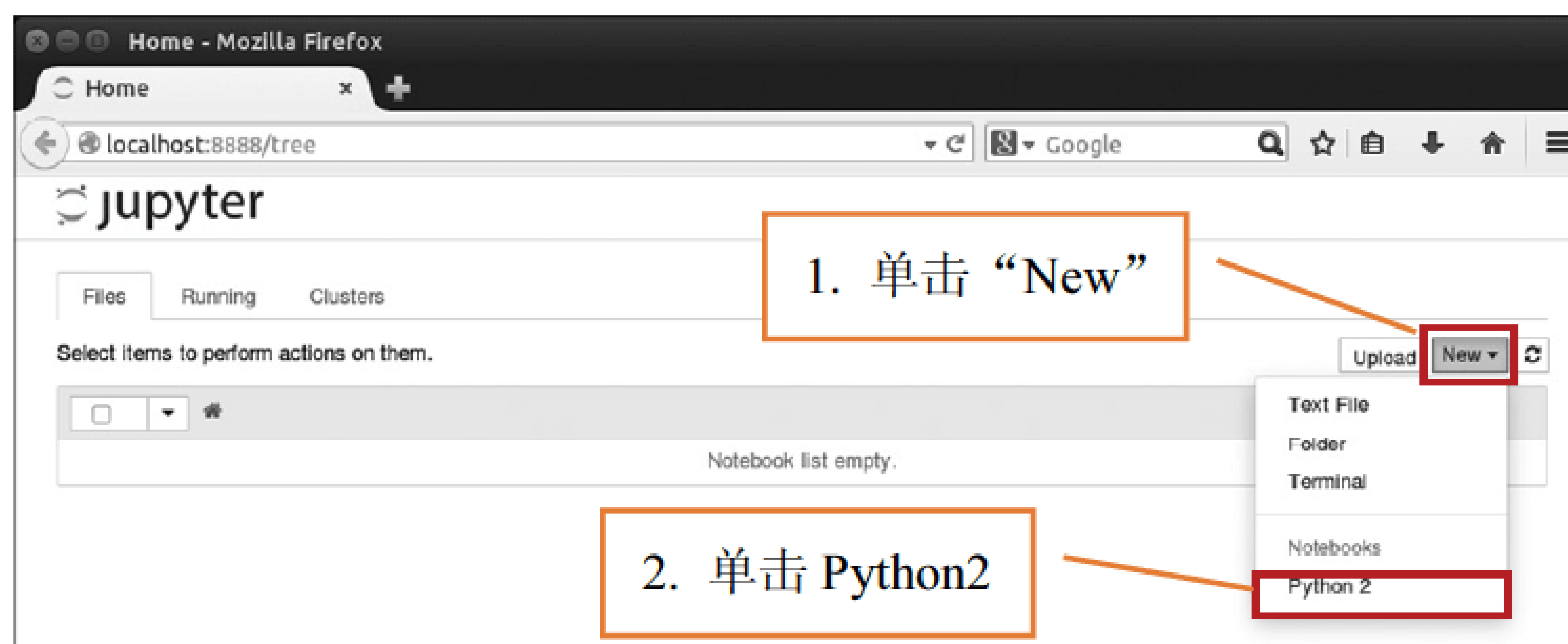


图 10-3 新建一个 Python Notebook

步骤 03 开始输入程序代码（如图 10-4）

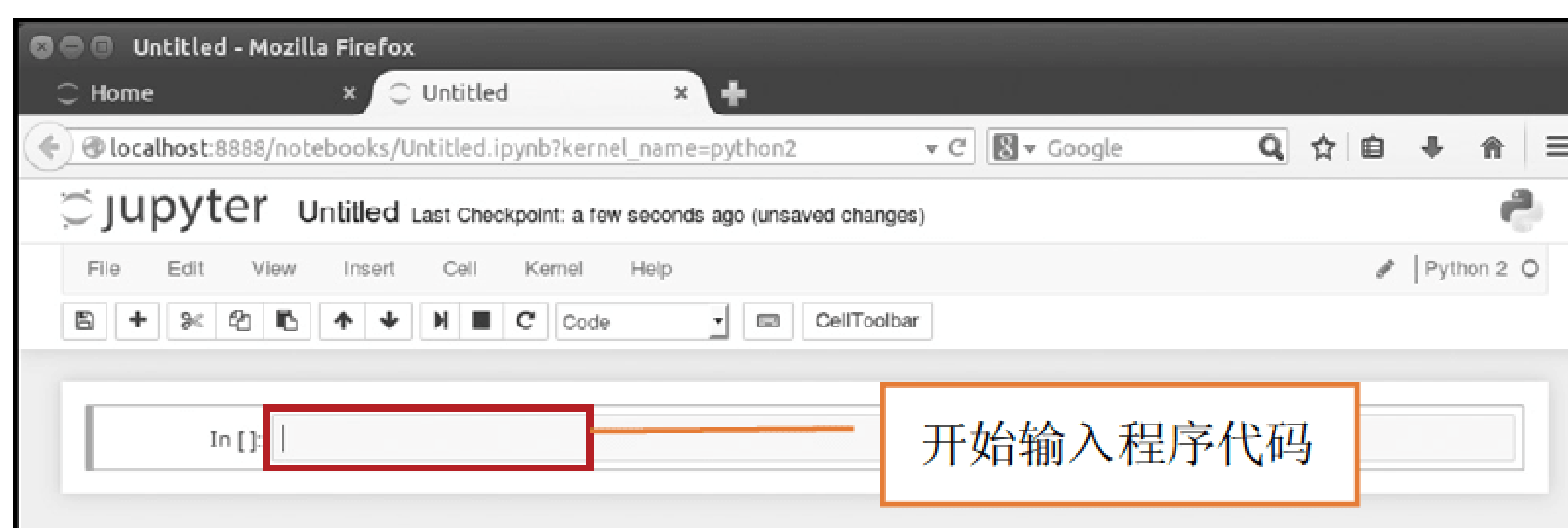
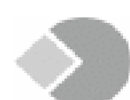


图 10-4 开始输入程序代码



10.3 基本 RDD“转换”运算

步骤 01 创建 intRDD

创建 RDD 最简单的方式就是使用 SparkContext 的 parallelize 方法，命令如下：

➢ **创建** intRDD

```
intRDD = sc.parallelize(List(3,1, 2, 5, 5))
```

先定义 intRDD，然后使用 parallelize 方法输入一个 List 的参数，以创建 intRDD。不过这也是一个“转换”运算，所以不会马上实际执行。

➢ intRDD **转换为** List

```
intRDD.collect()
```

intRDD 执行 collect()后会转换为 List。这是一个“动作”运算，所以会立刻执行，运行后的屏幕显示界面如图 10-5 所示。

```
In [1]:  intRDD = sc.parallelize([3,1, 2, 5, 5])
         intRDD.collect()
Out[1]:  [3, 1, 2, 5, 5]
```

图 10-5　创建 intRDD

步骤 02 创建 stringRDD

parallelize 方法除了可以创建 Int 的 RDD，也可以创建 String 的 RDD，命令如下：

➢ **创建** stringRDD

```
stringRDD = sc.parallelize(List("Apple","Orange","Banana","Grape","Apple"))
```

运行后，屏幕显示界面如图 10-6 所示。

```
In [2]:  stringRDD = sc.parallelize(["Apple", "Orange", "Banana","Grape","Apple"])
         stringRDD.collect()
Out[2]:  ['Apple', 'Orange', 'Banana', 'Grape', 'Apple']
```

1. 创建 StringRDD

2. 将 StringRDD 转换为 List

图 10-6　创建 stringRDD

步骤 03 map 运算介绍

map 运算可以通过传入的函数将每一个元素经过函数运算产生另外一个 RDD，如图 10-7 所示，RDD 通过传入的函数 addOne 将每一个元素加 1，从而产生另外一个 RDD。

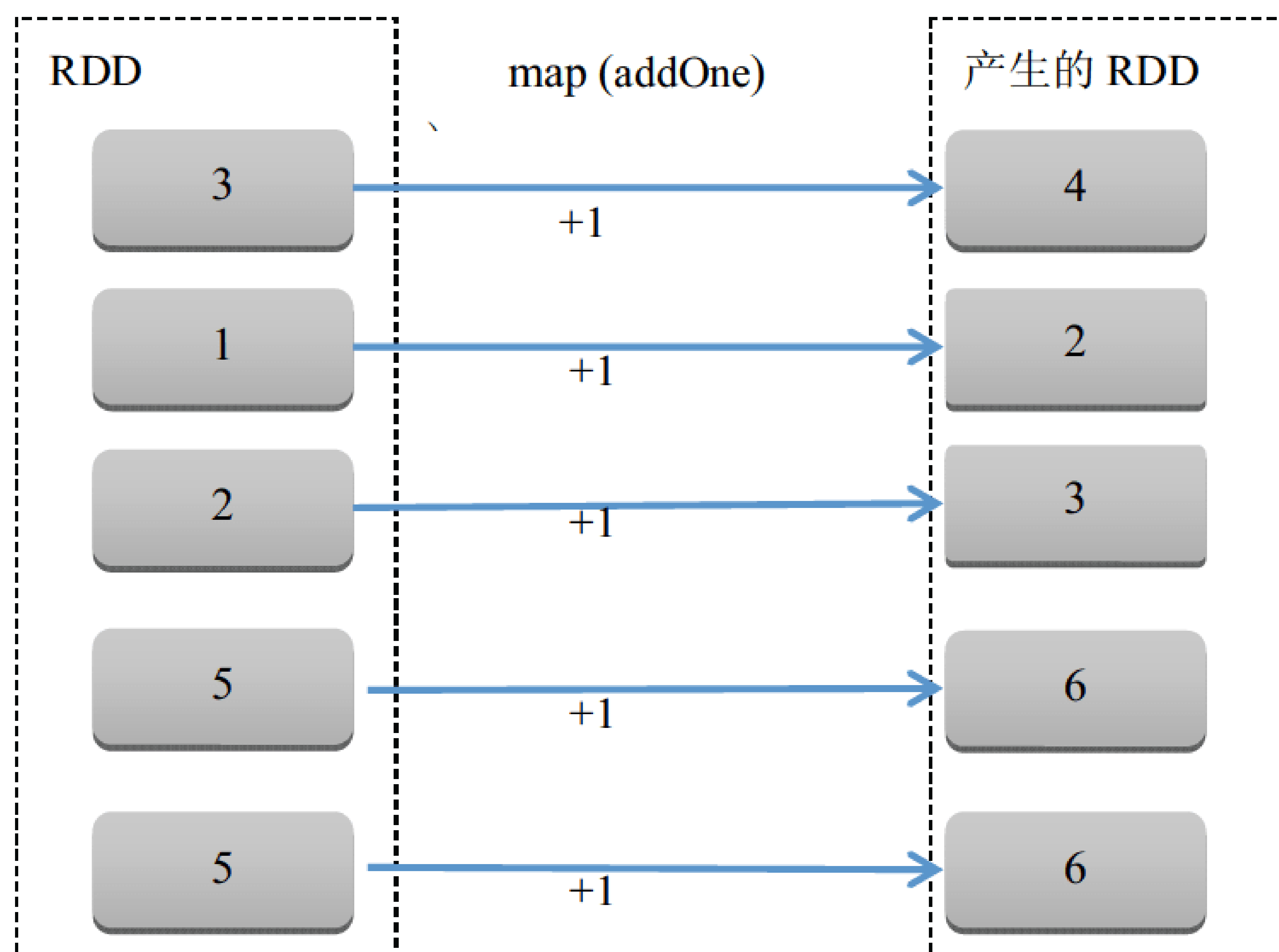


图 10-7　map 运算示意图

在 Spark 的 map 运算中，可以使用两种语句：具名函数和匿名函数。

步骤 04 map 运算使用具名函数

➢ **编写 addOne 函数**

先定义 addOne 函数并传入参数 x，此函数会将 x 加 1 再返回。

```
def addOne(x)
    return (x+1)
```

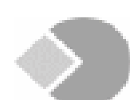
➢ **将函数名称 addOne 作为参数传入 map 函数**

```
intRDD.map(addOne).collect()
```

（1）将函数名称 addOne 作为参数传入 map 命令，map 命令会将每一个元素加 1，从而产生另外一个 RDD。

（2）因为 map 是一个"转换"运算，所以不会马上执行。为了方便示范，我们加上 collect() 这个"动作"运算使之立即执行。

在 iPython Notebook 输入指令，按 Shift + Enter 组合键运行，屏幕显示界面如图 10-8 所示。



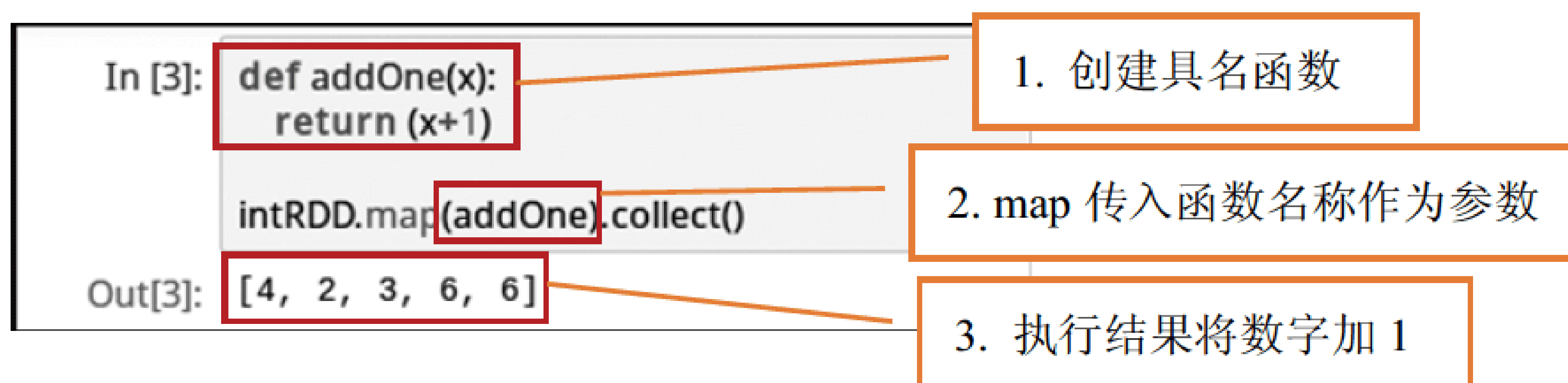


图 10-8　使用具名函数

以上运行结果是将原数组 [3,1,2, 5, 5]中的每一个数字都加 1 而变成的[4,2,3, 6, 6]。

步骤 05 map 运算使用匿名函数

map 运算可以使用更简单的语法，即匿名函数，命令如下：

```
intRDD.map(x => x + 1).collect()
```

如上述命令，map 会传入(x=>x+1)作为参数，这是 lambda 语句的匿名函数（anonymous functions）。其中，x 是传入参数，x+1 是要执行的命令，告诉 map 运算每一个元素都要加 1。在iPython Notebook 输入程序代码，按 Shift + Enter 组合键运行，屏幕显示界面如图 10-9 所示。

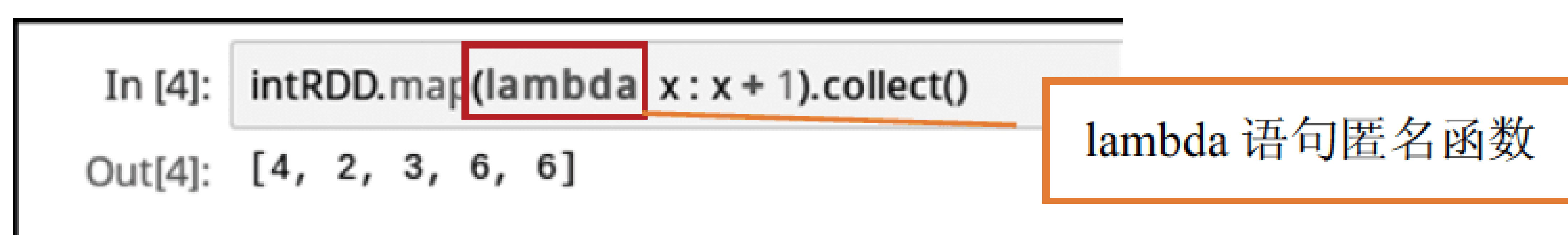


图 10-9　使用匿名函数

运行后，具名函数与匿名函数语句的执行结果完全相同，都是将 [3,1, 2, 5, 5]中的每一个数字都加 1 后变成 [4, 2, 3, 6, 6]。

步骤 06 具名函数或匿名函数的使用时机

何时使用具名函数与匿名函数呢？

（1）简单的功能使用匿名函数，复杂的功能使用具名函数。通常如果函数的功能很简单，例如本范例只是加 1，使用匿名函数会让程序语句简洁得多，而且让程序代码更易读。如果函数的功能过于复杂，就很难使用匿名函数来表达了，使用具名函数会比较清楚。

（2）重复使用的功能，使用具名函数。如果此函数在很多地方重复使用，建议使用具名函数。因为需要修改函数功能时，只需修改函数定义即可。如果是匿名函数，我们就必须修改好几个地方。

步骤 07 map 字符串运算

我们也可以针对字符串 RDD 执行 map 运算，如图 10-10 所示。

```
In [7]:  stringRDD.map(lambda x: "fruit:"+x).collect()
Out[7]:  ['fruit:Apple', 'fruit:Orange', 'fruit:Banana', 'fruit:Grape', 'fruit:Apple']
```

将 stringRDD 所有字符串元素前面加上 fruit: 来产生另一个 RDD

图 10-10　map 字符串运算

从以上界面可以看出，原来的 ["Apple", "Orange", "Banana","Grape","Apple"]全部加上了 fruit:，从而变成 ['fruit:Apple', 'fruit:Orange', 'fruit:Banana', 'fruit:Grape', 'fruit:Apple']。

步骤 08　filter 数字运算

filter 可以用于对 RDD 内每一个元素进行筛选，并产生另外的 RDD。

- **让 intRDD 筛选数字小于 3（见图 10-11）**

```
In [8]:  intRDD.filter(lambda x : x < 3).collect()
Out[8]:  [1, 2]
```

图 10-11　让 intRDD 筛选数字小于 3

原来的数组为 [3,1, 2, 5, 5]，因为筛选数字小于 3，所以结果是 [1 , 2]。

- **intRDD 筛选数字等于 3（见图 10-12）**

```
In [28]:  intRDD.filter(lambda x : x == 3).collect()
Out[28]:  [3]
```

图 10-12　筛选数字等于 3

- **intRDD 筛选数字在 1 到 5 之间（见图 10-13）**

```
In [26]:  intRDD.filter(lambda x : 1< x and x <5 ).collect()
Out[26]:  [3, 2]
```

图 10-13　筛选数字在 1 到 5 之间

原来的数组为 [3,1, 2, 5, 5]，因为筛选 1 到 5 之间的数字，所以结果是 [3 , 2]。

- **intRDD 筛选数字大于等于 5 或小于 3（见图 10-14）**

```
In [18]:  intRDD.filter(lambda x : x >=5 or x<3 ).collect()
Out[18]:  [1, 2, 5, 5]
```

图 10-14　筛选数字大于等于 5 或小于 3

原来的数组为[3,1, 2, 5, 5]，因为筛选大于等于 5 或小于 3 的数字，所以结果为[1, 2, 5, 5]。

步骤 09 filter 字符串运算

我们也可以针对字符串 RDD 执行 filter 运算。

> **筛选内含 ra 的字符串（见图 10-15）**

```
In [5]: stringRDD.filter(lambda x : "ra" in x ).collect()
Out[5]: ['Orange', 'Grape']
```

图 10-15 筛选内容 ra 的字符串

运行后，筛选结果为 Orange, Grape。

步骤 10 distinct 运算

distinct 运算会删除重复的元素。

> **删除 intRDD 重复的元素（见图 10-16）**

```
In [29]: intRDD.distinct().collect()
Out[29]: [1, 2, 3, 5]
```

图 10-16 删除 intRDD 重复的元素

intRDD 是 [3,1, 2, 5, 5] ，去除重复数字后为 [1,2,3,5]。

> **删除 stringRDD 重复的元素（见图 10-17）**

```
In [30]: stringRDD.distinct().collect()
Out[30]: ['Orange', 'Grape', 'Apple', 'Banana']
```

图 10-17 删除 stringRDD 重复的元素

stringRDD 是["Apple", "Orange", "Banana","Grape","Apple"]，去除重复元素后为 ["Orange", "Banana", "Grape","Apple"]。

步骤 11 randomSplit 运算

randomSplit 可以将整个集合元素以随机数的方式按照比例分为多个 RDD。

- 使用 randomSplit 将整个集合按照 4 : 6 的比例分为两个 RDD。

执行结果如图 10-18 所示。

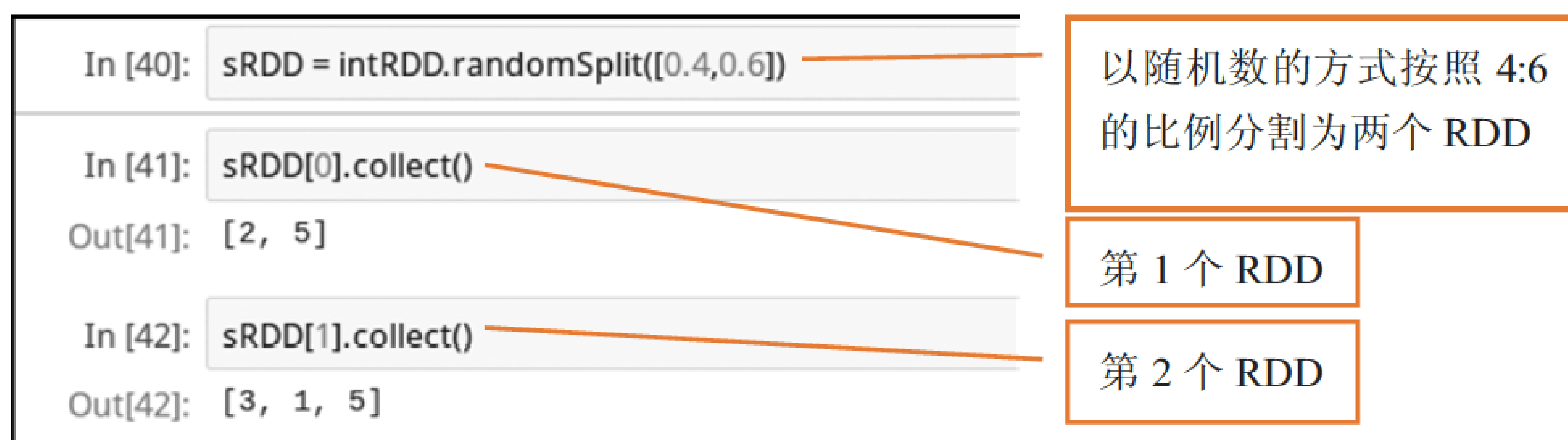


图 10-18 randomSplit 运算

步骤 12 groupBy 运算

groupBy 可以按照传入的匿名函数规则将数据分为多个 List。

➢ **使用 groupBy 运算将整个集合分成奇数与偶数（见图 10-19）**

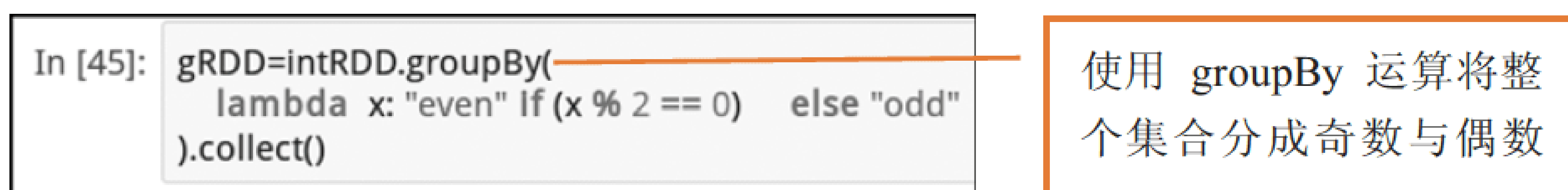


图 10-19 groupBy 运算

使用 groupBy 运算时，传入的匿名函数为(lambda x: "even" if (x % 2 == 0) else "odd")。以 if (x % 2 == 0) 判断传入的数字除以 2 的余数是否为 0，如果是就是偶数，否则为奇数。

➢ **查看奇数与偶数 List（见图 10-20）**

以上命令运行后会产生奇数与偶数两个 List，可以用下列指令查看。

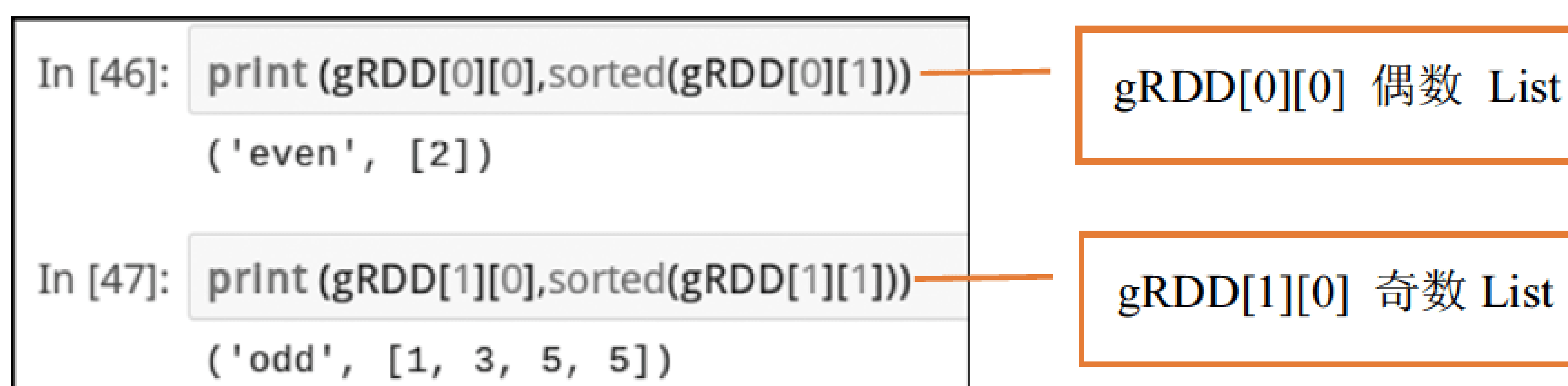


图 10-20 查看奇数与偶数 List

10.4 多个 RDD“转换”运算

RDD 也支持执行多个 RDD 的运算。

步骤 01 创建 3 个范例 RDD

为了示范多个 RDD 的运算，我们使用下列指令：



➢ **创建 3 个范例 RDD（见图 10-21）**

```
In [82]: intRDD1 = sc.parallelize([3, 1, 2, 5, 5])
         intRDD2 = sc.parallelize([5, 6])
         intRDD3 = sc.parallelize([2, 7])
```

图 10-21　创建 3 个范例 RDD

步骤 02　union 并集运算

可以使用下列指令将 intRDD1、intRDD2、intRDD3 进行并集运算。

➢ **使用 union 函数进行并集运算（见图 10-22）**

```
In [85]: intRDD1.union(intRDD2).union(intRDD3).collect()
Out[85]: [3, 1, 2, 5, 5, 6, 2, 7]
```

图 10-22　使用 union 函数进行并集运算

步骤 03　intersection 交集运算

可以使用下列指令进行运算：

➢ **将 intRDD1、intRDD2 进行交集运算（见图 10-23）**

```
In [87]: intRDD1.intersection(intRDD2).collect()
Out[87]: [5]
```

图 10-23　进行交集运算

intRDD1 是 [3, 1, 2, 5, 5]，与 intRDD2 [5, 6] 之间的重复元素只有 5，所以返回 [5]。

步骤 04　subtract 差集运算

可以使用下列指令进行运算：

➢ **进行差集运算（见图 10-24）**

```
In [54]: intRDD1.subtract(intRDD2).collect()
Out[54]: [1, 2, 3]
```

图 10-24　进行差集运算

intRDD1 是 [3, 1, 2, 5, 5]，扣除与 intRDD2 [5, 6] 重复的部分 5，结果是 [1,2,3]。

步骤 05　cartesian 笛卡儿乘积运算

可以使用下列指令进行运算：

➢ **进行 cartesian 笛卡儿乘积运算（见图 10-25）**

```
In [92]: print intRDD1.cartesian(intRDD2).collect()

[(3, 5), (3, 6), (1, 5), (1, 6), (2, 5), (2, 6), (5, 5),
 (5, 6), (5, 5), (5, 6)]
```

图 10-25　进行笛卡儿乘积运算

intRDD1 有 5 个元素、intRDD2 有 2 个元素，所以笛卡儿乘积运算后会产生 10（5*2）组数据。

10.5 基本“动作”运算

步骤 01 读取元素

可以使用下列指令读取 RDD 内的元素（见图 10-26），这是 Actions 运算，所以会马上执行。

```
In [96]: intRDD.first()
Out[96]: 3

In [94]: intRDD.take(2)
Out[94]: [3, 1]

In [31]: intRDD.takeOrdered(3)
Out[31]: [1, 2, 3]

In [32]: intRDD.takeOrdered(3,key=lambda x: -x)
Out[32]: [5, 5, 3]
```

取出第 1 项数据

取出第 2 项数据

从小到大排序，取出前 3 项

从大到小排序，取出前 3 项

图 10-26　读取元素

步骤 02 统计功能

还可以将 RDD 内的元素进行统计运算，参考图 10-27。



```
In [33]: intRDD.stats()
Out[33]: (count: 5, mean: 3.2, stdev: 1.6, max: 5.0, min: 1.0)

In [34]: intRDD.min()
Out[34]: 1

In [35]: intRDD.max()
Out[35]: 5

In [36]: intRDD.stdev()
Out[36]: 1.6000000000000001

In [37]: intRDD.count()
Out[37]: 5

In [38]: intRDD.sum()
Out[38]: 16

In [39]: intRDD.mean()
Out[39]: 3.2
```

统计　最小　最大　标准差　计数　总和　平均

图 10-27　统计功能

10.6 RDD Key-Value 基本“转换”运算

Spark RDD 支持键值（Key-Value）运算，Key-Value 运算也是 Map/Reduce 的基础，本节将介绍 RDD 键值的基本“转换”运算。

步骤 01　创建范例 Key-Value RDD

为了示范 RDD Key-Value 基本 Transformation 运算，先使用下列指令：

➢ **创建范例** Key-Value RDD（**见图** 10-28）

```
In [68]: kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
         kvRDD1.collect()
Out[68]: [(3, 4), (3, 6), (5, 6), (1, 2)]
```

图 10-28　创建范例 Key-Value RDD

第一个字段是 Key，第二个字段是 Value。例如，第一项数据(3,4)，3 是 key 值，4 是 value。

➢ **列出全部** key **值（第一个字段，见图** 10-29）

```
In [99]: kvRDD1.keys().collect()
Out[99]: [3, 3, 5, 1]
```

图 10-29　列出全部 key 值

➢ **列出 values 值（第二个字段，见图 10-30）**

```
In [101]: kvRDD1.values().collect()
Out[101]: [4, 6, 6, 2]
```

图 10-30　列出 values 值

步骤 02 使用 filter 筛选 key 运算

可以使用 filter 针对 key 筛选 RDD 内的元素，如下列指令：

➢ **使用 filter 筛选出 key<5（见图 10-31）**

```
In [104]: kvRDD1.filter(lambda keyValue: keyValue[0] < 5).collect()
Out[104]: [(3, 4), (3, 6), (1, 2)]
```

图 10-31　使用 filter 筛选出 key<5

上面 kvRDD1：(3, 4), (3, 6), (5, 6), (1, 2) 的 keyValue[0]（第一个字段是 Key 值）小于 5，共有 3 条数据(3,4),(3,6),(1,2)。

步骤 03 使用 filter 筛选 value 运算

可以使用 filter 针对 value 筛选出 RDD 内的元素，如下列指令：

➢ **使用 filter 筛选出 value<5（见图 10-32）**

```
In [105]: kvRDD1.filter(lambda keyValue: keyValue[1] < 5).collect()
Out[105]: [(3, 4), (1, 2)]
```

图 10-32　使用 filter 筛选出 value<5

以 kvRDD1 为例，(3, 4), (3, 6), (5, 6), (1, 2)的 keyValue[1]（第二个字段是 Value 值）小于 5，共有两项数据(3,4),(1,2)。

步骤 04 mapValues 运算

mapValues 运算可以针对 RDD 内每一组(Key,Value)进行运算，并且产生另外一个 RDD。

➢ **将 Value 的每一个值进行平方运算（见图 10-33）**

```
In [106]: kvRDD1.mapValues(lambda x : x * x).collect()
Out[106]: [(3, 16), (3, 36), (5, 36), (1, 4)]
```

图 10-33　将 Value 的每一个值进行平方运算

以 kvRDD1 为例，(3, 4), (3, 6), (5, 6), (1, 2)以 value（第二个字段）进行平方运算，也就是(3, 4*4), (3, 6*6), (5, 6*6), (1, 2*2)；结果是(3,16),(3,36),(5,36),(1,4)。



步骤 05　sortByKey 从小到大按照 key 排序

可以使用 sortByKey()按照 key 排序。传入参数默认值是 true，也就是从小到大排序（见图 10-34）。

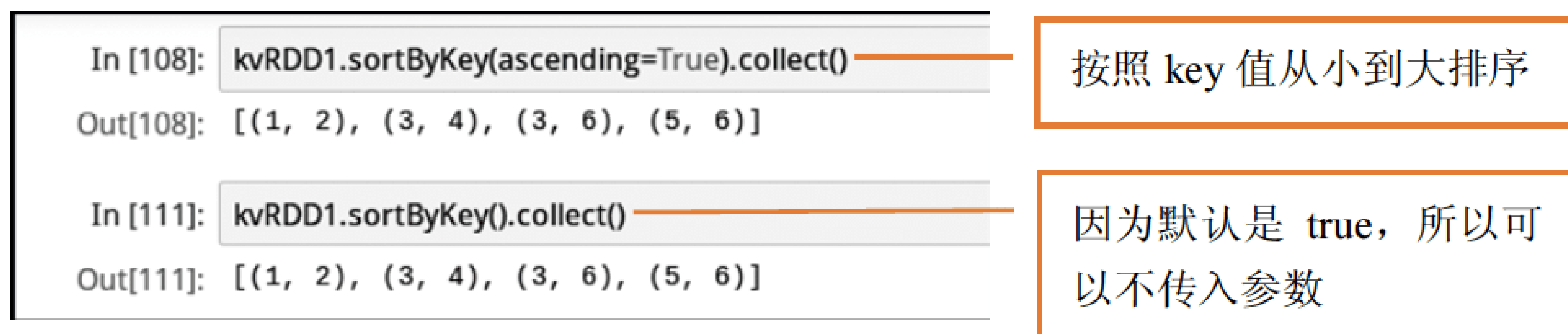


图 10-34　sortByKey 从小到大按照 key 排序

步骤 06　sortByKey 按照 key 值从大到小排序

按照 key 值从大到小排序，只需要在 sortByKey 传入参数 false 即可（参考图 10-35）。



图 10-35　按照 key 值从大到小排序

步骤 07　reduceByKey

另外一个很常见的功能是 reduceByKey，在 IPython NoteBook 输入图 10-36 所示的指令。



图 10-36　输入指令

reduceByKey 指令是按照 Key 值进行 reduce 运算，参考图 10-37 的说明。

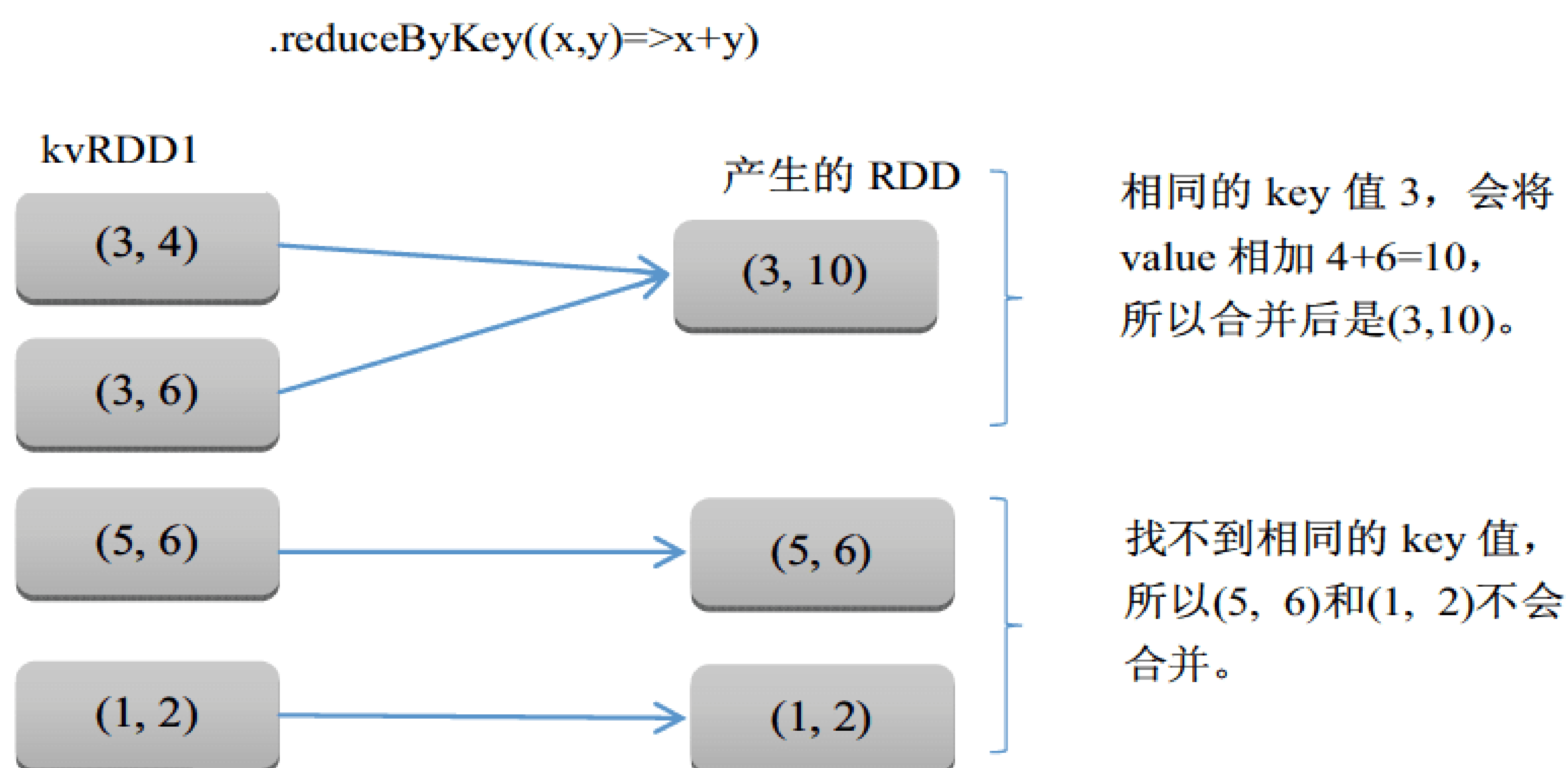


图 10-37　reduceByKey 命令运算范例的图解

kvRDD1.reduceByKey((x, y)=>x+y) 会将具有相同 Key 值的数据合并。合并的方式是按照

传入的匿名函数(x,y)=>x+y 相加，合并后产生另一个 RDD。以我们的测试数据为例：

（1）在 kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2)中，第一个字段是 key、第二个字段是 value。

（2）reduceByKey 会寻找相同的 key 合并，测试数据中 key 是 3，有 2 组数据(3, 4), (3, 6)。

（3）经过 reduceByKey 运算后，会将这 2 项数据(3, 4), (3, 6)合并为 1 条(3,10)，合并的方法是将 value 相加 4+6 = 10。

（4）剩下的(5, 6), (1, 2)没有相同的 key，所以不变。

10.7 多个 RDD Key-Value“转换”运算

步骤 01 Key-Value RDD 范例

为了示范多个 RDD Key-Value“转换”运算，先使用下列指令：

➢ **创建范例** Key-Value RDD（**见图** 10-38）

```
In [7]:   kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
          kvRDD2 = sc.parallelize([(3, 8)])

In [119]: kvRDD1.collect()
Out[119]: [(3, 4), (3, 6), (5, 6), (1, 2)]

In [120]: kvRDD2.collect()
Out[120]: [(3, 8)]
```

创建范例 Key-Value RDD

查看 kvRDD1

查看 kvRDD2

图 10-38　创建范例 Key-Value RDD

步骤 02 Key-Value RDD join 运算

join 运算可以将两个 RDD 按照相同的 key 值 join 起来，例如 kvRDD1 join kvRDD2，如图 10-39 所示。

```
In [122]: kvRDD1.join(kvRDD2).collect()
Out[122]: [(3, (4, 8)), (3, (6, 8))]
```

图 10-39　Key-Value RDD join 运算

以上程序代码说明如下：

（1）kvRDD1 与 kvRDD2 唯一相同的 key 值是 3。

（2）kvRDD1 是(3,4)、(3,6)，而 kvRDD2 是(3,8)，所以 join 的结果如图 10-40 所示。

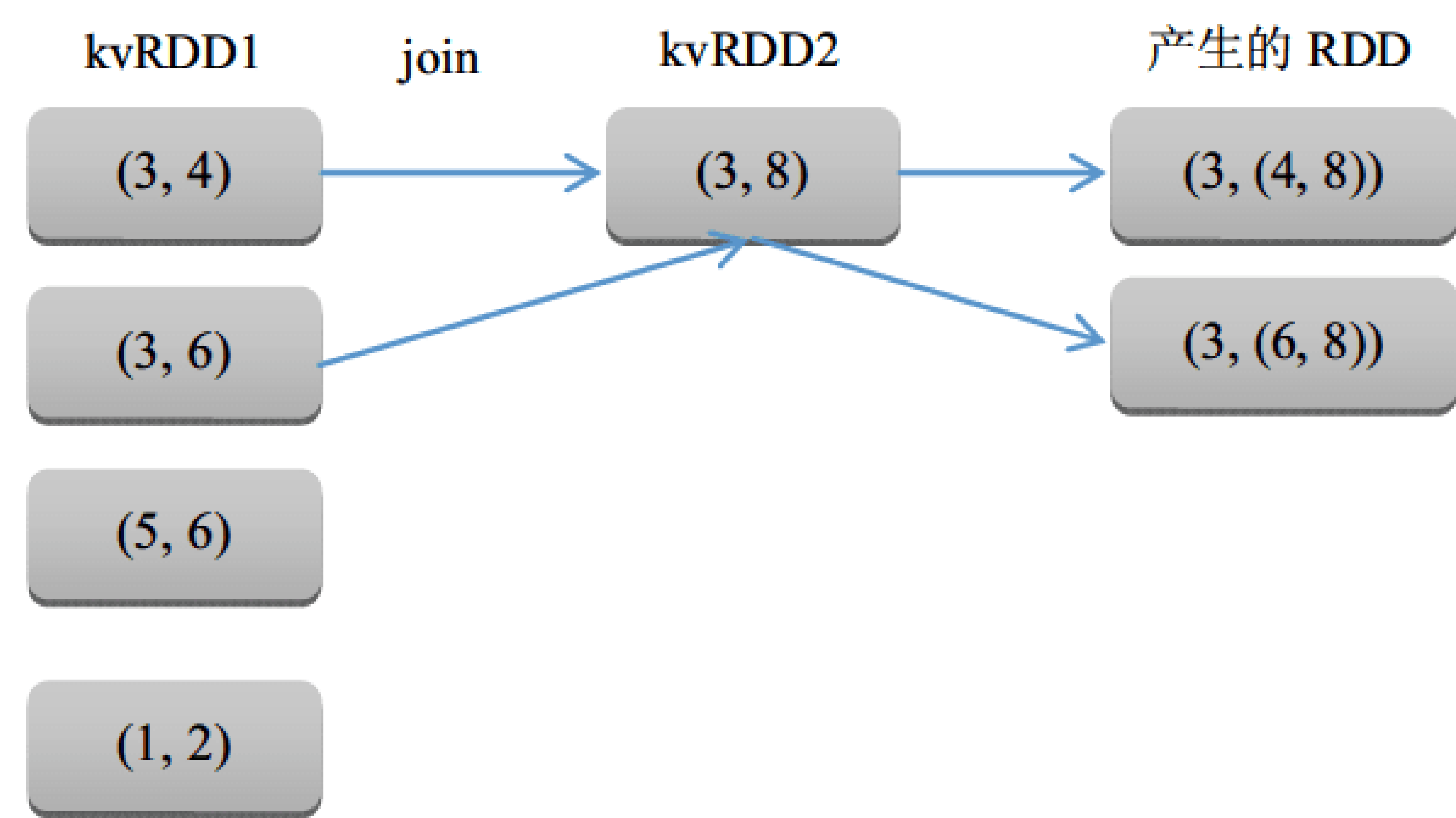


图 10-40 join 运算示意图

步骤03 Key-Value leftOuterJoin 运算

另外一种 Join 的方式是 Left Join（见图 10-41）。

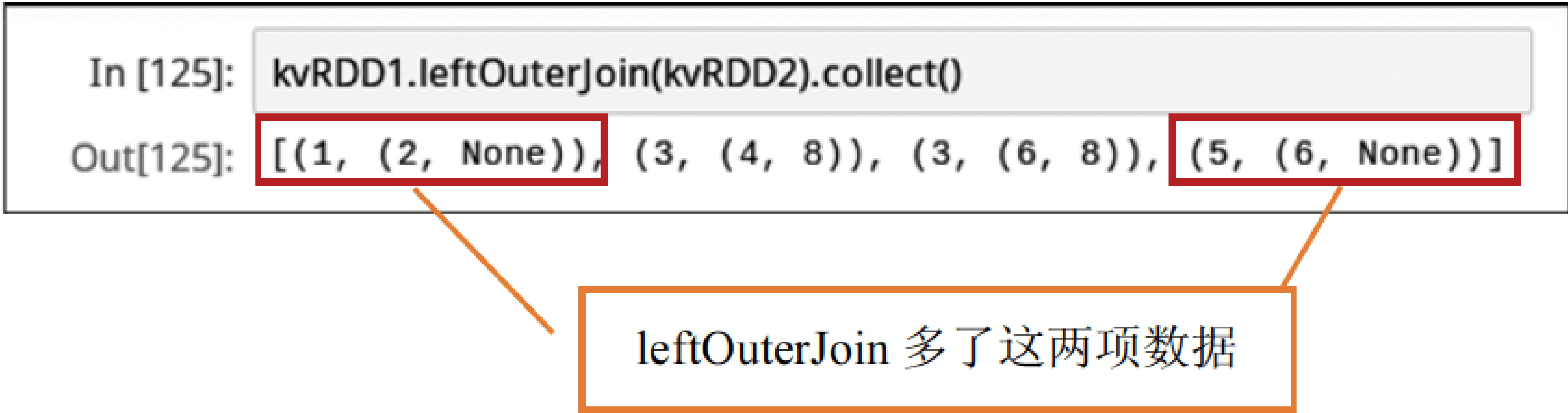


图 10-41 Left Join 运算

（1）leftOuterJoin 会从左边的集合（kvRDD1）对应到右边的集合（kvRDD2），并显示所有左边集合（kvRDD1）中的元素。

（2）如果 kvRDD1 的 key 值对应到 kvRDD2，就会显示相同的 key (3, (4, 8))、(3, (6, 8))。

（3）如果 kvRDD1 的 key 值对应不到 kvRDD2，就会显示 None(5,(6,None))、(1,(2,None))。

如图 10-42 所示。

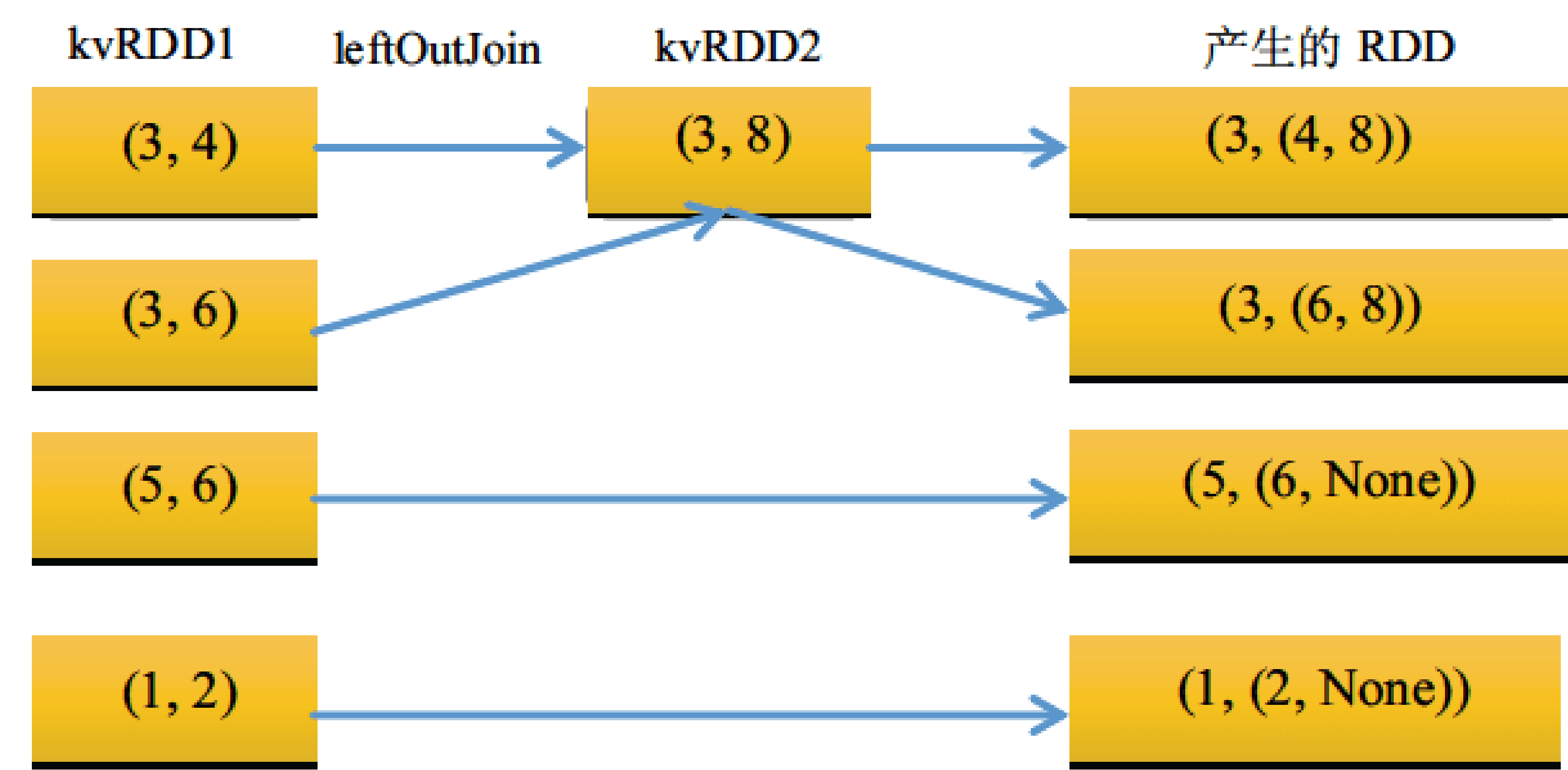


图 10-42 Left Join 运算示意图

步骤04 Key-Value RDD rightOuterJoin 运算

另外一种 Join 的方式是 Right Join（见图 10-43）。

```
In [127]: kvRDD1.rightOuterJoin(kvRDD2).collect()
Out[127]: [(3, (4, 8)), (3, (6, 8))]
```

图 10-43　Right Join 运算

说明如下：

（1）rightOuterJoin 会从右边的集合（kvRDD2）对应到左边的集合（kvRDD1），并显示所有右边集合（kvRDD2）中的元素。

（2）如果 kvRDD2 的 key 值对应到 kvRDD1，就会显示重复的 key(3, (4, 8))、(3, (6, 8))。

如图 10-44 所示。

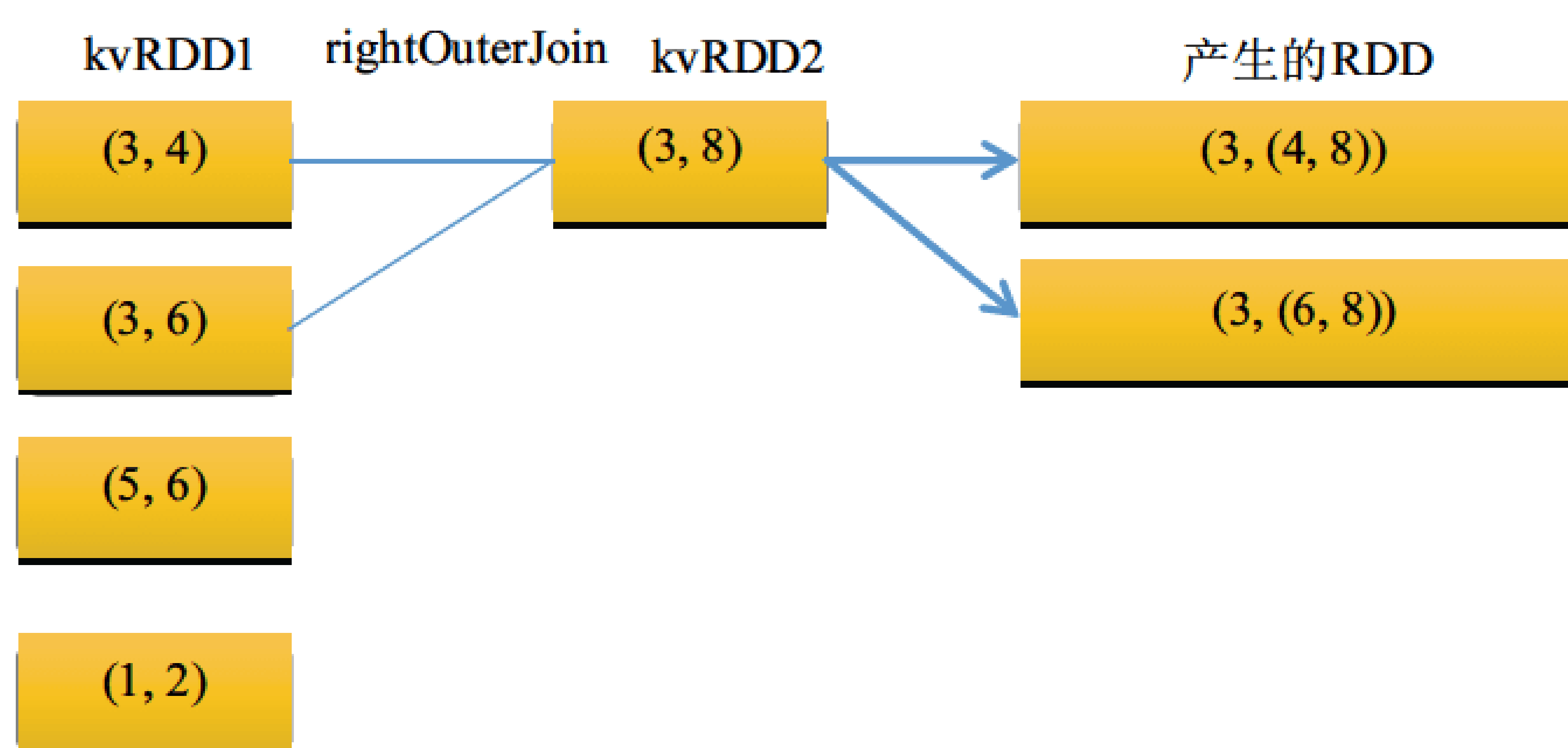


图 10-44　Key-Value RDD rightOuterJoin 运算范例的图解

步骤 05　Key-Value subtractByKey 运算

subtractByKey 运算会删除相同 key 值的数据。例如，kvRDD1 subtract kvRDD2，如图 10-45 所示。

```
In [128]: kvRDD1.subtractByKey(kvRDD2).collect()
Out[128]: [(1, 2), (5, 6)]
```

图 10-45　subtractByKey 运算

以上程序代码 kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2)，删除与 kvRDD2 具有相同 key 值 3 的项，所以只剩下(5, 6)与(1,2)，如图 10-46 所示。

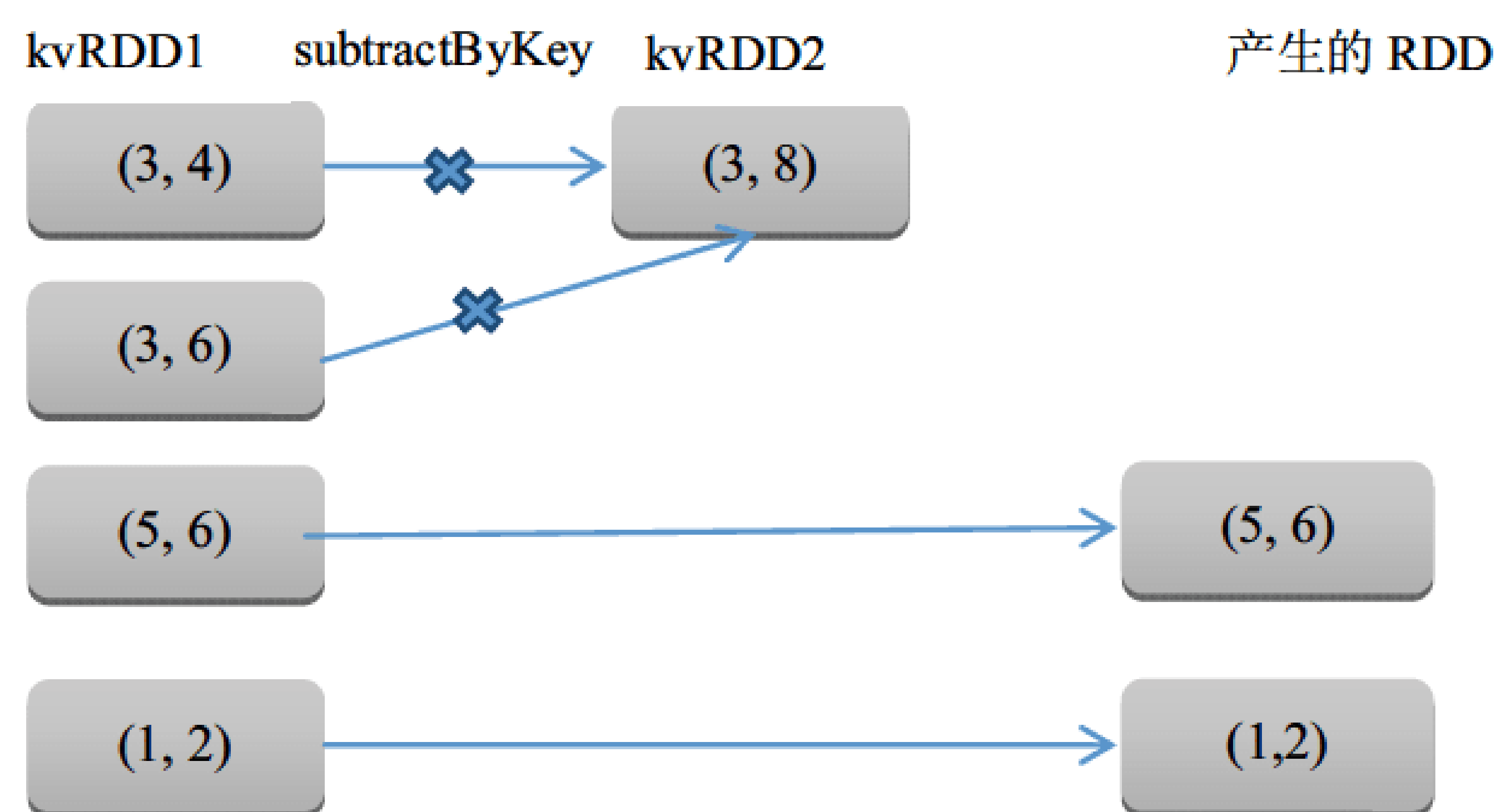


图 10-46　Key-Value subtractByKey 运算范例的图解

10.8 Key-Value“动作”运算

步骤 01 Key-Value first 运算

可以使用图 10-47 中的指令读取 RDD 部分的数据。

```
In [129]:  kvRDD1.first()
Out[129]:  (3, 4)

In [131]:  kvRDD1.take(2)
Out[131]:  [(3, 4), (3, 6)]
```

获取第一项数据

获取前两项数据

图 10-47　使用 first 运算读取数据

步骤 02 读取第一项数据的元素

使用 kvRDD1.first() 获取第一项数据并存储在 kvFirst 变量中。在 Python 中可以使用 kvFirst[0] 获取第 1 个元素、kvFirst[1] 取得第 2 个元素……依此类推，可参考图 10-48。

```
In [7]:    kvFirst=kvRDD1.first()
           kvFirst
Out[7]:    (3, 4)

In [134]:  kvFirst[0]
Out[134]:  3

In [135]:  kvFirst[1]
Out[135]:  4
```

获取第 1 项数据

获取第 1 项数据的第 1 个元素，也就是 Key 值

获取第 1 项数据的第 2 个元素，也就是 Value 值

图 10-48　读取第一项数据的元素

步骤 03 计算 RDD 中每一个 Key 值的项数

例如，kvRDD1 数据(3, 4), (3, 6), (5, 6), (1, 2)，key 值为 3 的有两项数据，其余 key 值（1 和 5）都只有一项数据，所以结果为：1->1 项、3->2 项、5->1 项。

运行后屏幕显示界面如图 10-49 所示。

```
In [8]: kvRDD1.countByKey()
Out[8]: defaultdict(int, {1: 1, 3: 2, 5: 1})
```

图 10-49 计算 RDD 中每一个 Key 值的项数

步骤 04 collectAsMap 创建 Key-Value 的字典

Python 字典数据类型就好像真实的字典，可以用“字”查询“字的解说”。其中，“字”就是 key，“字的解说”就是 value。

我们可以使用 collectAsMap 创建 Key-Value 的字典。例如，kvRDD1 数据 (3, 4), (3, 6), (5,6), (1, 2) 能帮我们产生字典 dict(1:2, 3:6, 5:6)，不过 Key=3 有两个值 4 和 6，系统只能自动对应到其中的值 6，也就是 3:6。

运行后屏幕显示界面如图 10-50 所示。

```
In [91]: KV=kvRDD1.collectAsMap()
         KV
Out[91]: {1: 2, 3: 6, 5: 6}

In [92]: type(KV)
Out[92]: dict
```


图 10-50 利用 collectAsMap 创建 Key-Value 字典

步骤 05 使用对照表转换数据

字典建立后，使用字典转换数据，运行后屏幕显示界面如图 10-51 所示。

```
In [141]: KV[3]
Out[141]: 6

In [142]: KV[1]
Out[142]: 2
```


图 10-51 使用对照表转换数据

步骤 06 Key-Value lookup 运算

可以使用 lookup 输入 key 值来查找 value 值，以 kvRDD1((3, 4), (3, 6), (5, 6), (1, 2))为例，可参考图 10-52。

```
In [144]: kvRDD1.lookup(3)
Out[144]: [4, 6]

In [130]: kvRDD1.lookup(5)
Out[130]: [6]
```

输入 key 值 3，找到两项(3,4),(3, 6)，所以 value 值是 4 和 6

输入 key 值 5，找到 1 项 (5, 6)，value 值是 6

图 10-52　lookup 运算

10.9 Broadcast 广播变量

接下来，我们将介绍 Shared variable 共享变量。共享变量可用于节省内存与运行时间，提升并行处理时的执行效率。共享变量包括 Broadcast 和 accumulator。

- Broadcast 广播变量——本节会介绍。
- accumulator 累加器——下一节会介绍。

步骤 01 不使用 Broadcast 广播变量的范例

我们先看不使用 Broadcast 广播变量的范例，这个范例很简单，说明如下：

先创建水果编号与名称对照表，然后使用此对照表将水果编号转换为水果名称。

➢ 创建 kvFruit

这是水果编号与名称的 Key-Value RDD（见图 10-53）。

```
In [1]: kvFruit = sc.parallelize([(1, "apple"), (2, "orange"), (3, "banana"), (4, "grape")])
```

图 10-53　创建 kvFruit

➢ 创建 fruitMap 字典

使用 collectAsMap 创建 fruitMap 字典（水果编号与名称对照表），可参考图 10-54。

```
In [2]: fruitMap=kvFruit.collectAsMap()
        print "字典："+str(fruitMap)

字典：{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

图 10-54　创建 fruitMap 字典

➢ 创建 fruitIds（见图 10-55）

```
In [3]: fruitIds=sc.parallelize([2,4,1,3])
        print " 水果编号："+str(fruitIds.collect()),

水果编号：[2, 4, 1, 3]
```

图 10-55　创建 fruitIds

➢ **使用 fruitMap 字典进行转换（见图 10-56）**

```
In [4]: print " 使用字典进行转换 =>"
        fruitNames= fruitIds.map(lambda x : fruitMap[x]).collect()
        print " 水果名称 : "+str(fruitNames)

使用字典进行转换 =>
水果名称 : ['orange', 'grape', 'apple', 'banana']
```

图 10-56　使用 fruitMap 字典进行转换

上面的范例执行起来虽然没有任何问题，但是在并行处理中每执行一次转换都必须将 fruitIds 与 fruitMap 传送到 Worker Node，才能够执行转换，如图 10-57 所示。如果字典 fruitMap（对照表）很大，而且需要转换的 fruitIds 水果编号 RDD 也很大，就会耗费很多内存与时间。

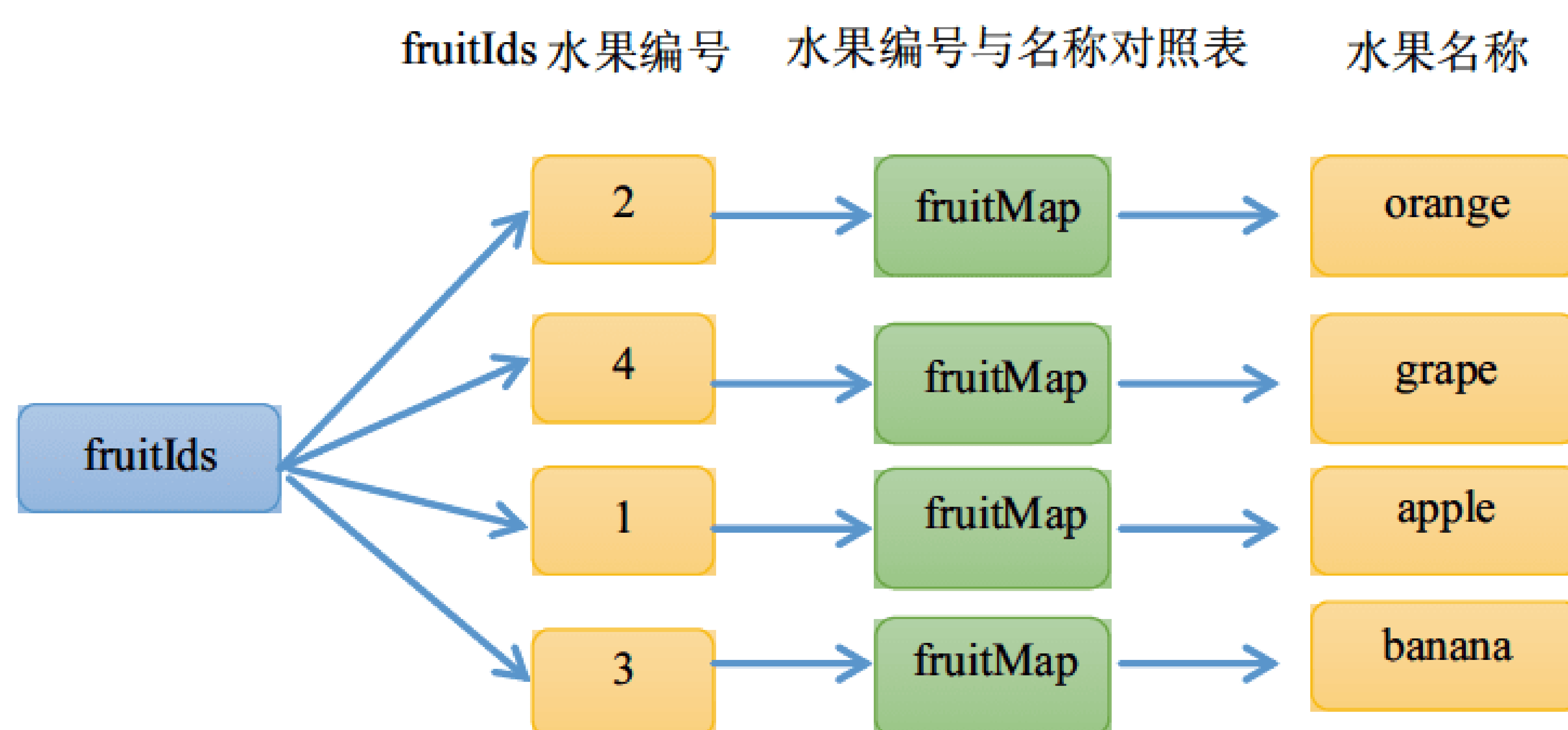


图 10-57　将水果编号转换为水果名称范例的示意图

为了解决这个问题，就必须使用 Broadcast 广播变量。

步骤 02　使用 Broadcast 广播变量的范例

Broadcast 广播变量使用规则如下：

（1）可以使用 SparkContext.broadcast([初始值])创建。
（2）使用.value 的方法来读取广播变量的值。
（3）Broadcast 广播变量被创建后不能修改。

下列使用 Broadcast 广播变量的范例与之前的范例类似，不同之处是使用 sc.broadcast 传入 fruitMap 作为参数，创建 bcFruitMap 广播变量，使用 bcFruitMap.value(x) 广播变量转换为 fruitNames 水果名称。

下列程序代码与之前步骤 1 所示的程序代码类似，不同之处是改用 bcFruitMap 广播变量。

➢ **创建 kvFruit**

这是水果编号与名称的 Key-Value RDD（见图 10-58）。

```
In [1]: kvFruit = sc.parallelize([(1, "apple"), (2, "orange"), (3, "banana"), (4, "grape")])
```

图 10-58　创建 kvFruit

➢ **创建 fruitMap 字典（见图 10-59）**

使用 collectAsMap 创建 fruitMap 字典（水果编号与名称对照表）。

```
In [2]: fruitMap=kvFruit.collectAsMap()
        print "对照表：" +str(fruitMap)

        对照表：{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

图 10-59　创建 fruitMap 字典

➢ **将 fruitMap 字典转换为 bcFruitMap 广播变量**

使用 sc.broadcast 传入 fruitMap 参数，创建 bcFruitMap 广播变量（见图 10-60）。

```
In [6]: fruitMap=kvFruit.collectAsMap()
        bcFruitMap=sc.broadcast(fruitMap)
        print "字典：" +str(fruitMap)

        字典：{1: 'apple', 2: 'orange', 3: 'banana', 4: 'grape'}
```

使用 sc.broadcast 传入 fruitMap 参数，创建 bcFruitMap 广播变量

图 10-60　将 fruitMap 字典转换为 bcFruitMap 广播变量

➢ **创建 fruitIds（见图 10-61）**

```
In [3]: fruitIds=sc.parallelize([2,4,1,3])
        print "水果编号：" +str(fruitIds.collect()),

        水果编号：[2, 4, 1, 3]
```

图 10-61　创建 fruitIds

➢ **使用 bcFruitMap.value 字典进行转换（见图 10-62）**

```
In [8]: print "使用Broadcast 广播变量字典进行转换 =>"
        fruitNames= fruitIds.map(lambda x : bcFruitMap.value[x]).collect()
        print "水果名称：" +str(fruitNames)

        使用Broadcast 广播变量字典进行进行转换 =>
        水果名称：['orange', 'grape', 'apple', 'banana']
```

使用 bcFruitMap.value[x]广播变量将 fruitIds 转换为 fruitNames

图 10-62　使用 bcFruitMap.value 字典进行转换

执行的结果与之前步骤 1 范例相同。在并行处理中，bcFruitMap 广播变量会传送到 Worker Node 机器，并且存储在内存中，如图 10-63 所示。后续在此 Worker Node 都可以使用这个 bcFruitMap 广播变量执行转换，这样就可以节省很多内存与传送时间。

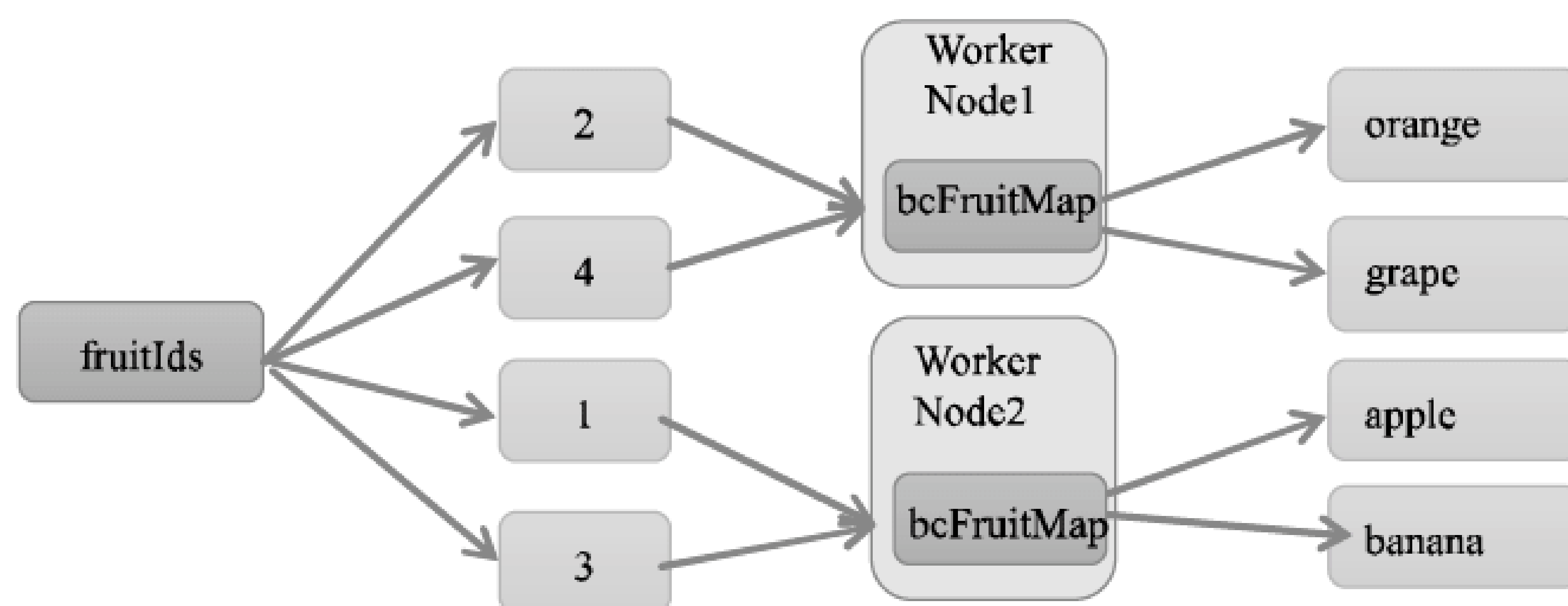


图 10-63　使用 Broadcast 广播变量执行范例的图解

10.10 accumulator 累加器

步骤 01 accumulator 累加器的介绍

计算总和是 MapReduce 常用的运算。为了方便并行处理，Spark 特别提供了 accumulator 累加器共享变量（Shared variable），使用规则如下：

- accumulator 累加器可以使用 SparkContext.accumulator([初始值])来创建。
- 使用 .add() 进行累加。
- 在 task 中，例如 foreach 循环中，不能读取累加器的值。
- 只有驱动程序，也就是循环外，才可以使用.value 来读取累加器的值。

步骤 02 accumulator 累加器范例

接下来，我们使用下列范例计算 RDD 的求和、计数。

➢ **创建范例 RDD（见图 10-64）**

```
In [9]: intRDD = sc.parallelize([3,1, 2, 5, 5])
```

图 10-64　创建范例 RDD

➢ **创建 total 累加器，初始值使用 0.0，所以是 Double 的类型（见图 10-65）**

```
In [10]: total = sc.accumulator(0.0)
```

图 10-65　创建 total 累加器

➢ **创建 num 累加器，初始值使用 0，所以是 Int 的类型（见图 10-66）**

```
In [11]: num = sc.accumulator(0)
```

图 10-66　创建 num 累加器

➢ **使用 foreach 传入参数 i，针对每一项数据执行**

total 累加 intRDD 元素的值、num 累加 intRDD 元素的数量（见图 10-67）。

```
In [12]: intRDD.foreach(lambda i: [total.add(i) , num.add(1)])
```

图 10-67　使用 foreach 进行累加

➢ **计算平均=求和 / 计数，并显示总和、数量（见图 10-68）**

```
In [13]: avg = total.value / num.value
         print("total="+str(total.value)+", num="+str(num.value)+", avg="+str(avg))

total=16.0, num=5, avg=3.2
```

图 10-68　计算机平均值并显示总和、数量

10.11　RDD Persistence 持久化

Spark RDD 持久化机制可以用于将需要重复运算的 RDD 存储在内存中，以便大幅提升运算效率。

Spark RDD 持久化使用方法如下：

- RDD.persist（存储等级）——可以指定存储等级，默认是 MEMORY_ONLY，也就是存储在内存中。
- RDD.unpersist()——取消持久化。

持久化存储等级说明如下：

➢ MEMORY_ONLY

这是默认选项。存储 RDD 的方式是以 Java 对象反串行化在 JVM 内存中，如果 RDD 太大无法完全存储在内存，多余的 RDD partitions 不会缓存在内存中，而是需要时再重新计算。

➢ MEMORY_AND_DISK

存储 RDD 的方式是以 Java 对象反串行化在 JVM 内存中。如果 RDD 太大就无法完全存储在内存，会将多余的 RDD partitions 存储在硬盘中，需要时再从硬盘读取。

➢ MEMORY_ONLY_SER

与 MEMORY_ONLY 类似，但是存储 RDD 以 Java 对象串行化，因为需要再进行反串行化才能使用，所以会多使用 CPU 的计算资源，但是比较省内存的存储空间。多余的 RDD partitions 不会缓存在内存中，而是需要时再重新计算。

➢ MEMORY_AND_DISK_SER

与 MEMORY_ONLY_SER 类似，会将多余的 RDD partitions 存储在硬盘中，需要时再从硬盘读取。

➢ DISK_ONLY

存储 RDD 在硬盘上。

➢ MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.

与上列相对应的存储选项一样，但是每一个 RDD partitions 都复制到两个节点。

步骤 01 使用 RDD.persist()持久化（见图 10-69）

```
In [8]:  intRddMemory = sc.parallelize([3,1, 2, 5, 5])
Out[8]:  ParallelCollectionRDD[5] at parallelize at PythonRDD.scala:423

In [22]: intRddMemory.persist()
Out[22]: ParallelCollectionRDD[5] at parallelize at PythonRDD.scala

In [9]:  intRddMemory.is_cached
Out[9]:  True
```

1. 创建 intRddMemory
2 将 intRddMemory 持久化
3. 查看是否已经 cached（缓存）

图 10-69 持久化

步骤 02 使用 RDD.unpersist()取消持久化（见图 10-70）

```
In [10]:  intRddMemory.unpersist()
Out[10]:  ParallelCollectionRDD[5] at parallelize at PythonRDD.scala:423

In [11]:  intRddMemory.is_cached
Out[11]:  False
```

1. 取消持久化
2. 查看是否已经取消缓存
3. False 代表已经取消

图 10-70 取消持久化

步骤 03 RDD.persist 设置存储等级范例（见图 10-71）

```
In [23]:  intRddMemoryAndDisk = sc.parallelize([3,1, 2, 5, 5])

In [24]:  intRddMemoryAndDisk.persist(StorageLevel.MEMORY_AND_DISK)
Out[24]:  ParallelCollectionRDD[12] at parallelize at PythonRDD.scala:423

In [25]:  intRddMemoryAndDisk.is_cached
Out[25]:  True
```

1. 创建 RDD
2. 设置存储等级
3. 已经 cached（缓存）

图 10-71 设置存储等级



10.12 使用 Spark 创建 WordCount

本章到目前为止，我们学到了 map 与 reduceByKey。接下来，我们将使用 map 与 reduceByKey 编写 Spark 版本的 WordCount。

步骤 01 创建测试文件

在“终端”程序中的提示符后输入下列命令：

➢ **创建 WordCount 的数据目录**

```
mkdir -p  ~/pythonwork/ipynotebook/data
```

➢ **切换至 WordCount 的数据目录**

```
cd ~/pythonwork/ipynotebook/data
```

➢ **编辑 test.txt**

```
gedit test.txt
```

运行后屏幕显示界面如图 10-72 所示。

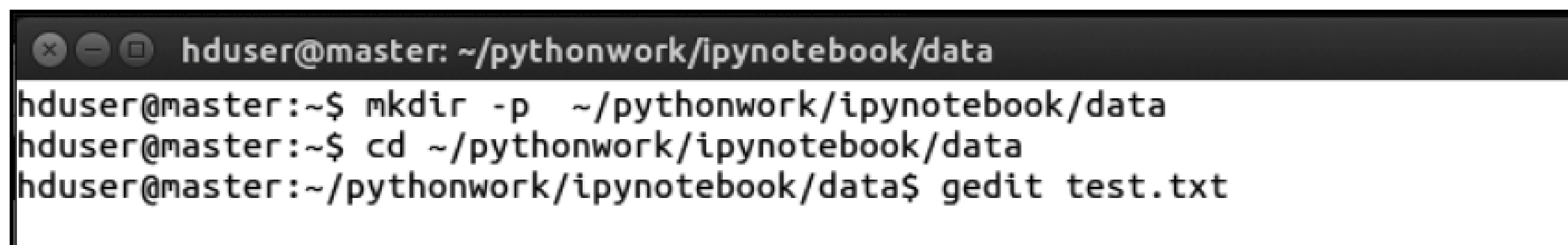


图 10-72　创建测试文件

输入后按 Enter 键就会打开 test.txt，屏幕显示界面如图 10-73 所示。



图 10-73　在 test.txt 文件中输入内容

输入内容后单击“保存”按钮，再关闭该文件。

步骤 02 执行 WordCount spark 命令

在 Spark 运行 WordCount 就是如此简单，只需要 4 行命令。与第 7 章 Hadoop MapReduce 介绍的 wordCount.Java 比较，可以发现程序代码简单很多，而且容易理解。在 pyspark 输入下

列命令，这些命令的详细用法，我们下一节会详细解说。

➢ **读取文本文件**

```
textFile=sc.textFile("data/test.txt")
```

➢ **使用 flatMap 空格符分隔单词，并读取每个单词**

```
stringRDD=textFile.flatMap(lambda line => line.split(" "))
```

➢ **通过 map reduce 计算每一个单词出现的次数**

```
countsRDD=stringRDD.map(lambda word : (word, 1)).reduceByKey(lambda x, y : x+y)
```

➢ **保存计算结果**

```
countsRDD.saveAsTextFile("data/output")
```

运行后屏幕显示界面如图 10-74 所示。

```
In [120]:  textFile = sc.textFile("data/test.txt")
           stringRDD=textFile.flatMap(lambda line : line.split(" "))
           countsRDD = stringRDD.map(lambda word : (word, 1)) \
                                .reduceByKey(lambda x,y : x+y)
           countsRDD.saveAsTextFile("data/output")
```

图 10-74　执行 WordCount spark 命令

步骤 03 查看执行目录

因为之前在第 10.2 节我们进入 IPython Notebook 时的工作路径是 ~/pythonwork/ipynotebook/，所以执行的结果会存储在工作路径下的 data/output 目录。下面我们使用相对路径查看文件，执行的结果会存储在/home/hduser/pythonwork/ipynotebook/data/output 目录。

➢ **查看 data 目录**

可以在 IPython Notebook 查看目录，请使用 ll 命令来查看 data 目录，但是必须加上“%”符号，告诉 IPython Notebook 这是一个 shell 命令。

```
%ll data
```

➢ **查看 output 目录**

在 iPython Notebook 输入下列命令切换到 output 目录，查看输出目录。

```
%ll data/output
```

➢ **查看 part-00000 输出文件**

在 iPython Notebook 输入下列命令，查看输出文件内容：

```
%cat data/output/part-00000
```

运行后屏幕显示界面如图 10-75 所示。

```
In [31]: %ll data
总用量 4222
-rwxrwxrwx 1 root 4318308  1月 23  2012 free-zipcode-database-Primary.csv*
drwxrwxrwx 1 root    4096  5月 30 21:40 output/
-rwxrwxrwx 1 root      38  3月 12 23:48 test.txt*

In [32]: %ll data/output
总用量 1
-rwxrwxrwx 1 root 58  5月 30 21:40 part-00000*
-rwxrwxrwx 1 root  0  5月 30 21:40 _SUCCESS*

In [33]: %cat data/output/part-00000
(u'Orange', 1)
(u'Grape', 2)
(u'Apple', 2)
(u'Banana', 1)
```

产生的 output 目录

产生的输出文件

输出文件的内容

图 10-75　查看执行目录

从执行结果可以看到 Grape 与 Apple 都出现了两次，而 Orange 与 Banana 都只出现了一次。

步骤 04　发生目录已经存在的错误

如果我们再次输入下列指令，output 目录已经存在，无法写入数据，就会产生错误，出现如图 10-76 所示的错误信息。

```
In [2]: textFile = sc.textFile("data/test.txt")
        stringRDD=textFile.flatMap(lambda line : line.split(" "))
        countsRDD = stringRDD.map(lambda word : (word, 1)) \
                             .reduceByKey(lambda x,y : x+y)
        countsRDD.saveAsTextFile("data/output")

---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-2-3f7f9da0b4cc> in <module>()
      2 stringRDD=textFile.flatMap(lambda line : line.split(" "))
      3 countsRDD = stringRDD.map(lambda word : (word, 1))
x+y)
----> 4 countsRDD.saveAsTextFile("data/output")
```

输出目录已经存在，发生错误

图 10-76　发生目录已经存在的错误

步骤 05　删除输出目录（见图 10-77）

```
In [28]: %rm -R data/output
```

删除输出目录

图 10-77　删除输出目录

删除后，我们再执行一次 WordCount 就不会发生错误了。

10.13 Spark WordCount 详细解说

上一节已经示范了 Spark 的 WordCount 程序，因为 map reduce 原理很重要，所以在本节将详细介绍 WordCount 的每一条命令。图 10-78 为示意图。

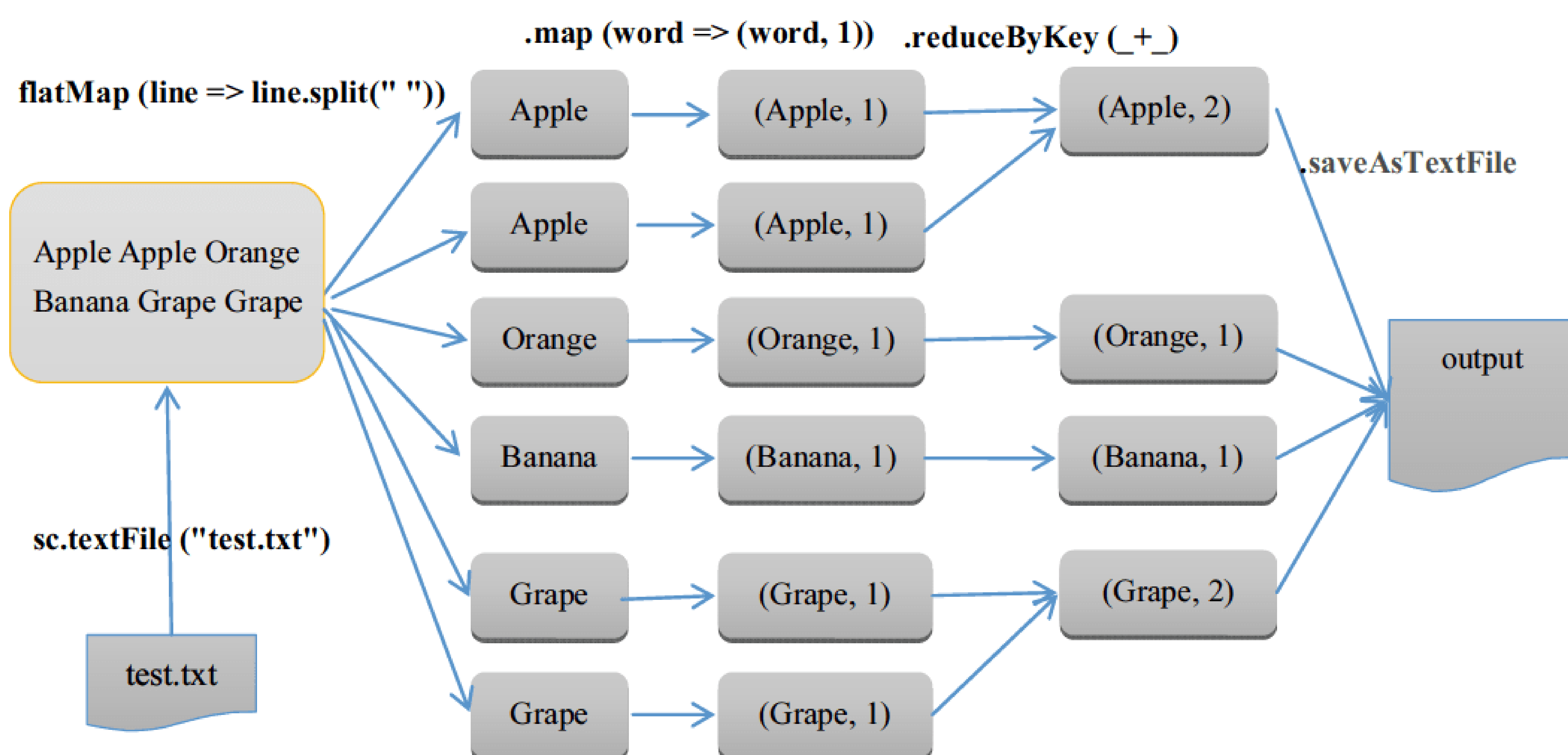


图 10-78 Spark WordCount 运行流程示意图

如图 10-78 所示，首先用 sc.textFile 读取文件，经过 flatMap 产生 6 项 RDD，再经过 map 产生 (word,1)，然后用.reduceByKey 进行统计，最后使用 saveAsTextFile 存储。

步骤 01 sc.textFile 读取本地文件

首先，输入下列指令读取本地文本文件。因为 sc.textFile 是一个转换运算，所以不会马上实际执行。可以使用下 textFile.collect()显示出文本文件内容，运行后屏幕显示界面如图 10-79 所示。

```
In [7]: textFile = sc.textFile("data/test.txt")
        textFile.collect()
Out[7]: [u'Apple Apple Orange', u'Banana Grape Grape']
```

图 10-79 通过 sc.textFile 读取本地文件

步骤 02 flatMap 读取每一个单词

文本文件以空格符分隔单词，我们可以使用 flatMap 指令取出每一个文字，并编写 stringRDD。因为之前的指令是一个“转换”运算，不会马上实际执行，所以使用 collect 将结果转换为 List，就会立刻实际执行，如图 10-80 所示。



```
In [6]: stringRDD=textFile.flatMap(lambda line : line.split(" "))
        stringRDD.collect()
Out[6]: [u'Apple', u'Apple', u'Orange', u'Banana', u'Grape', u'Grape']
```

图 10-80　通过 flatMap 读取每一个单词

步骤 03 flatMap 与 map 的差异

在这里要特别注意，在上一步骤以空格符分隔单词创建 stringRDD 时，必须要使用 flatMap，不能使用 map，因为 flatMap 与 map 功能不同。我们测试下列两条命令，就可以了解这两条命令的差异了。

➢ map 命令

map 产生的 List 是分层的。第一层 List 是文本文件的每一行、第二层 List 是每一行内的英文单词。执行后的屏幕显示界面如图 10-81 所示。

```
In [67]: stringRDD=textFile.map(lambda line : line.split(" "))
         stringRDD.collect()
Out[67]: [[u'Apple', u'Apple', u'Orange'], [u'Banana', u'Grape', u'Grape']]
```

具有分层

图 10-81　map 命令产生的 List 具有分层

➢ flatMap 命令

flat 有平坦的意思，也就是说 flatMap 产生的 List 会将所有分层去掉，结果为 List(Apple, Apple, Orange, Banana, Grape, Grape)，执行后的屏幕显示界面如图 10-82 所示。

```
In [9]: stringRDD=textFile.flatMap(lambda line : line.split(" "))
        stringRDD.collect()
Out[9]: [u'Apple', u'Apple', u'Orange', u'Banana', u'Grape', u'Grape']
```

无分层

图 10-82　flatMap 命令产生的 List 无分层

步骤 04 map 创建 Key-Value Pair

接下来，我们使用 map 来创建 Key-Value Pair。因为有了 Key-Value Pair，就很容易进行并行运算了，如图 10-83 所示。

```
In [10]: stringRDD.map(lambda word : (word, 1)).collect()
Out[10]: [(u'Apple', 1),
          (u'Apple', 1),
          (u'Orange', 1),
          (u'Banana', 1),
          (u'Grape', 1),
          (u'Grape', 1)]
```

图 10-83　使用 map 创建 key-Value Pair

（1）使用 map 命令将 stringRDD 每一个英文单词转换为 Key-Value。

（2）其中 Key 值是每一个英文单词、Value 值都是 1。

（3）因为 map 是“转换”运算，所以不会马上实际运行。为了立刻运行看到结果，可以加上“行动”运算.foreach(println)输出每一条数据。

步骤05 map 加 reduceByKey

接下来，使用 map 加上 reduceByKey 完成 Map/Reduce 功能。

reduceByKey 传入匿名函数(lambda x,y : x+y)，此匿名函数会将相同的 key 值相加。因为 reduceByKey 是“转换”运算，所以不会马上运行。加上“动作”运算 .collect()，转换为 List，就会马上执行，如图 10-84 所示。

```
In [76]: stringRDD.map(lambda word : (word, 1)).reduceByKey(lambda x,y : x+y).collect()
Out[76]: [(u'Orange', 1), (u'Grape', 2), (u'Apple', 2), (u'Banana', 1)]
```

图 10-84 使用 map 加 reduceByKey 完成 Map/Reduce 功能

执行后，我们可以看到 Grape 与 Apple 都出现了两次，而 Orange 与 Banana 都只出现了 1 次。

10.14 结论

本章我们使用 IPython NoteBook 介绍了 Spark 的基本功能 RDD，还用 WordCount 作为 Spark Map/Reduce 统计文章中单词的字数。IPython NoteBook 虽然使用很方便，但是有时我们希望能够创建一个完整的 Python Spark 程序，可以重复执行数据分析。要开发 Python Spark 应用程序，最好是使用集成开发环境（IDE），这样更有效率，所以下一章将介绍如何使用 eclipse 配合 pyDev 开发 Python Spark 应用程序。

第 11 章

Python Spark的集成开发环境

之前我们介绍 pyspark 与 IPython Notebook 执行 Python Spark 命令，优点是具有交互性，输入命令后可以立刻看到响应。如果我们希望能够成交一个完整的 Python Spark 程序，就可以重复执行数据分析。要开发 Python Spark 应用程序，最好是有集成开发环境 Integrated Development Environment（IDE），这样更有效率。

本章我们将介绍使用 eclipse 集成开发环境（IDE）来开发 Spark 应用程序。eclipse 是很受欢迎的跨平台、开放源码、集成开发环境（IDE）。eclipse 是一个开发框架，最初用于开发 Java 语言的程序，并且可以通过外挂模块的支持，使其成为其他程序设计语言的开发工具，例如 C++、Python、PHP、Scala 等。

Python 集成开发环境很多，本书介绍使用 eclipse，并且选择使用 Scala IDE eclipse 集成开发环境，这是因为 Spark 本身就是 Scala 开发的，使用 Spark 很有可能会用到 scala 语言，所以我们希望能够在一个开发环境中开发 Scala 与 Python。不过本书主要是介绍 Python 开发 Spark 应用程序，想要学习以 Scala 开发 Spark 应用程序，可以参考笔者的另外一本著作《Hadoop+Spark 大数据巨量分析与机器学习整合开发实战》。

➢ 集成开发环境的好处

使用集成开发环境的好处很多，举例如下：

（1）Code Completion：自动完成程序代码。如图 11-1 所示，输入 sc 后按下“.”就会弹出此对象能够使用的 method。有了这个功能，就不需要强记命令了，非常方便。

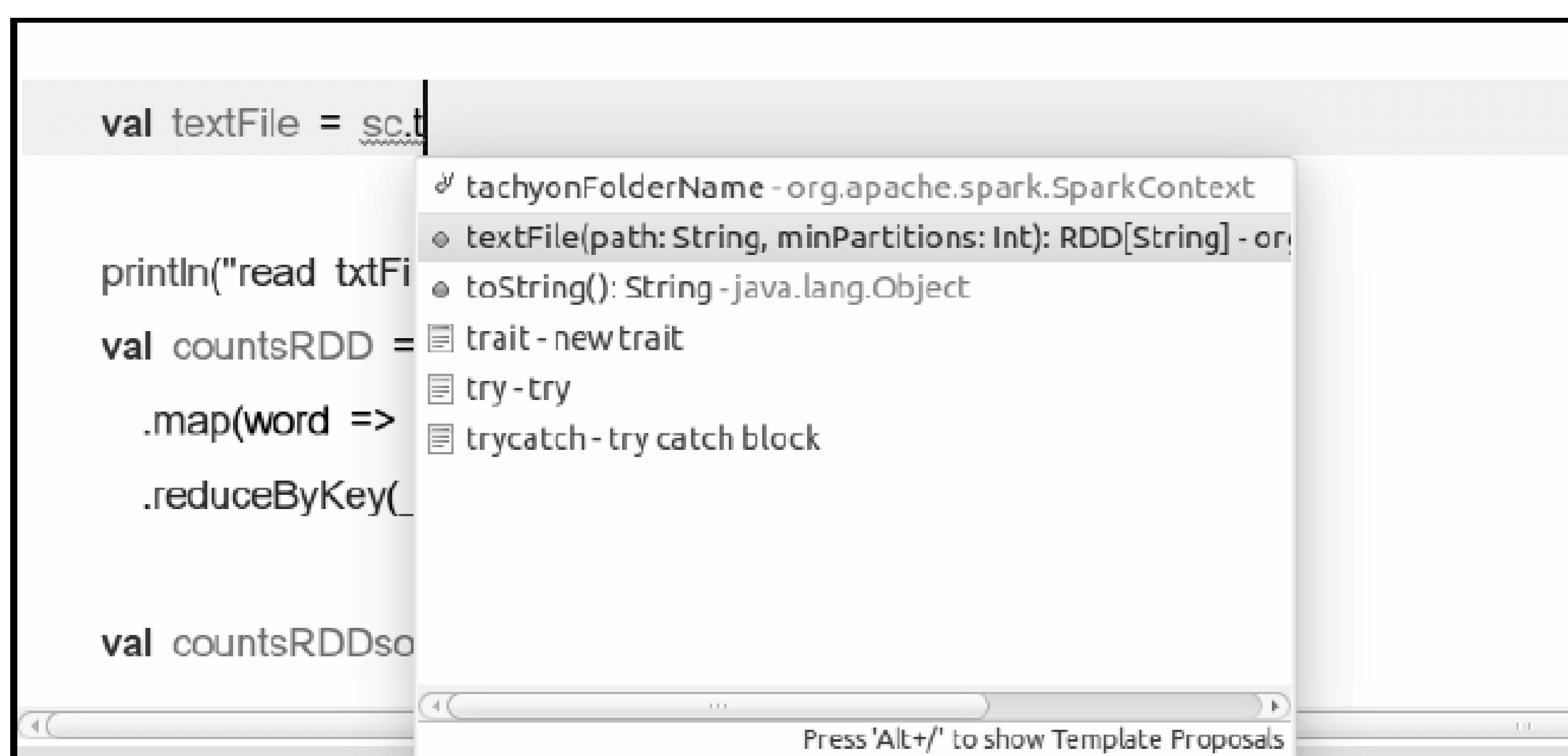


图 11-1　自动完成程序代码

（2）Semantic Highlighting：关键词、对象、变量、字符串、注释等都用不同的颜色标注出来，让程序代码更易读，如图 11-2 所示。

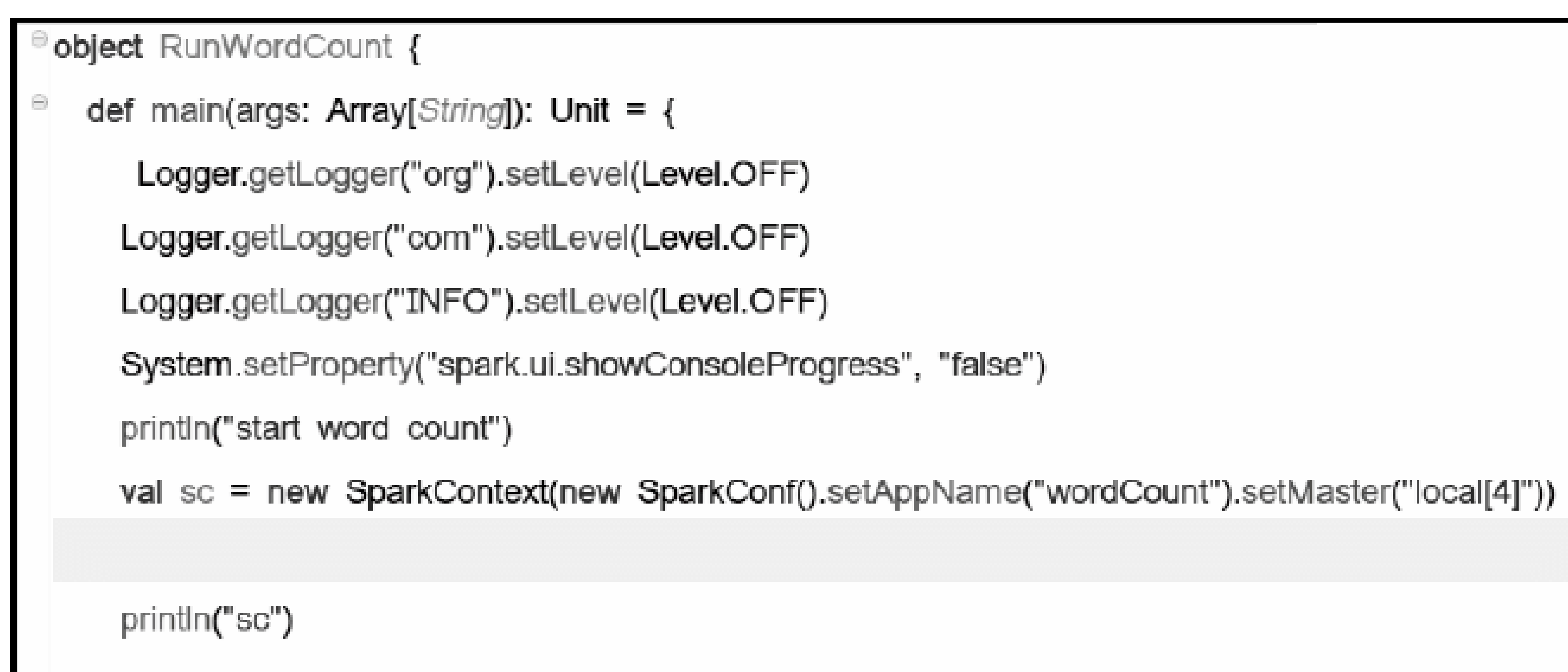


图 11-2　用不同颜色标注各项

（3）程序代码的编辑、编译、调试、运行都在同一个环境中，大幅度提升了软件开发的效率，如图 11-3 所示。

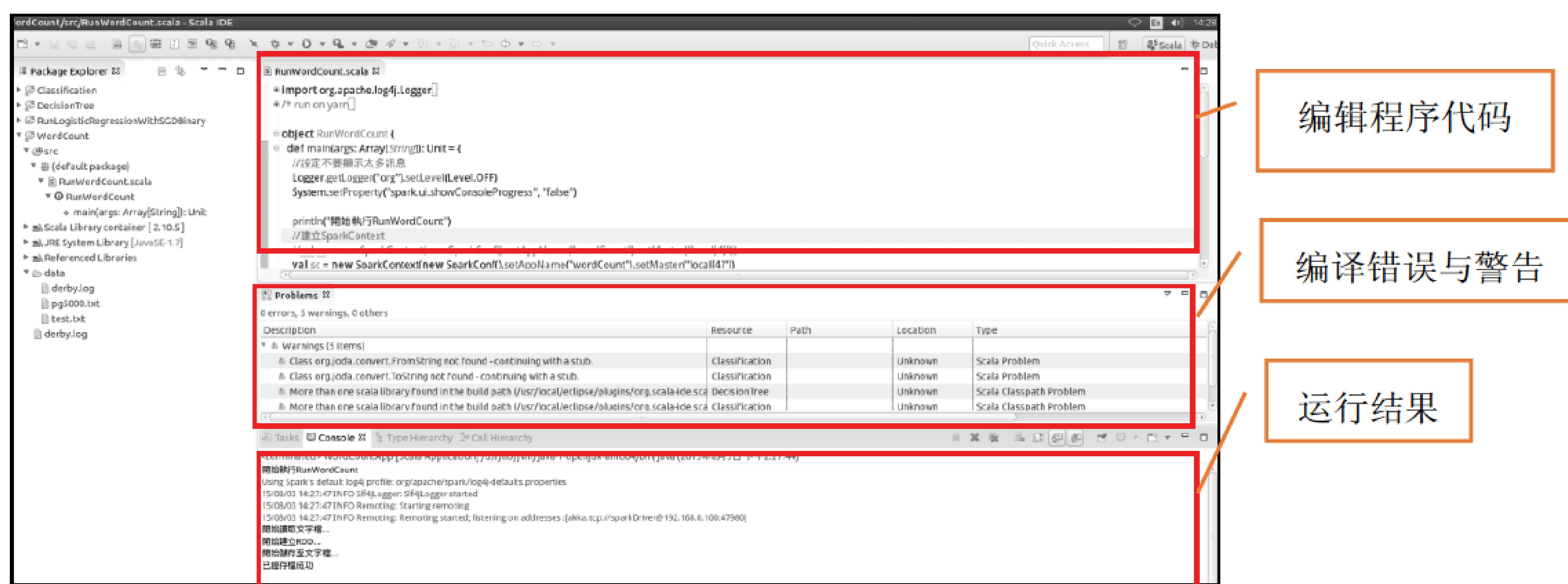


图 11-3　程序代码的编辑、编译、运行都在同一个环境中

➢ **Spark Python 集成开发环境的安装步骤（见表 11-1）**

表 11-1　Spark Python 集成开发环境的安装步骤

顺序	安装步骤	说明
1	下载与安装 eclipse Scala IDE	选择使用 Scala IDE eclipse 版本，我们希望能够在一个开发环境中开发 scala 与 Python
2	安装 PyDev	为了能够在 eclipse 开发 Python，必须安装 PyDev 套件
3	设置字符串替代变量	字符串替代变量的作用是代表某一个字符串：我们将设置 3 个字符串替代变量 SPARK_HOME 、HADOOP_CONF_DIR、PYSPARK_PYTHON，后续的设置过程需使用这些字符串替代变量
4	PyDev 设置 Python 链接库路径	必须在 PyDev 设置指定 Python 2.7 链接库的路径
5	PyDev 设置 anaconda2 链接库路径	必须在 PyDev 设置指定 anaconda2 链接库的路径
6	PyDev 设置 Spark Python 链接库	当我们执行 Spark Python 程序时必须引用 Spark Python 程序库，所以必须设置 Spark Python 链接库
7	PyDev 设置环境变量	当我们在 eclipse 执行 Spark Python 程序时，必须设置两个环境变量 SPARK_HOME 与 HADOOP_CONF_DIR。我们将使用之前设置的字符串替代变量设置环境变量，简化设置过程

➢ **本章命令整理**

本章使用的命令已经整理在本书的博客文章中，在练习安装时可以复制博客文章中的命令，然后粘贴到“终端”程序中。这样既可节省打字的时间，也不用担心打错字（无法在 VirtualBox 虚拟机的 Ubuntu“终端”程序中执行复制/粘贴操作时，可参考第 3.9 节的说明，设置好 VirtualBox 的共享剪贴板）。本书的博客网址为：

```
http://blog.sina.com.cn/hadoopsparkbook
```

11.1 下载与安装 eclipse Scala IDE

下载 Scala IDE eclipse 集成开发环境，将其安装在 master 服务器上。

步骤01 浏览 Scala IDE 网页

在 master 服务器中，启动浏览器，输入下列网址，进入 Scala IDE 官网，如图 11-4 所示。

```
http://scala-ide.org/
```



图 11-4 浏览 Scala IDE 网页

步骤02 Scala IDE 下载页面（见图 11-5）

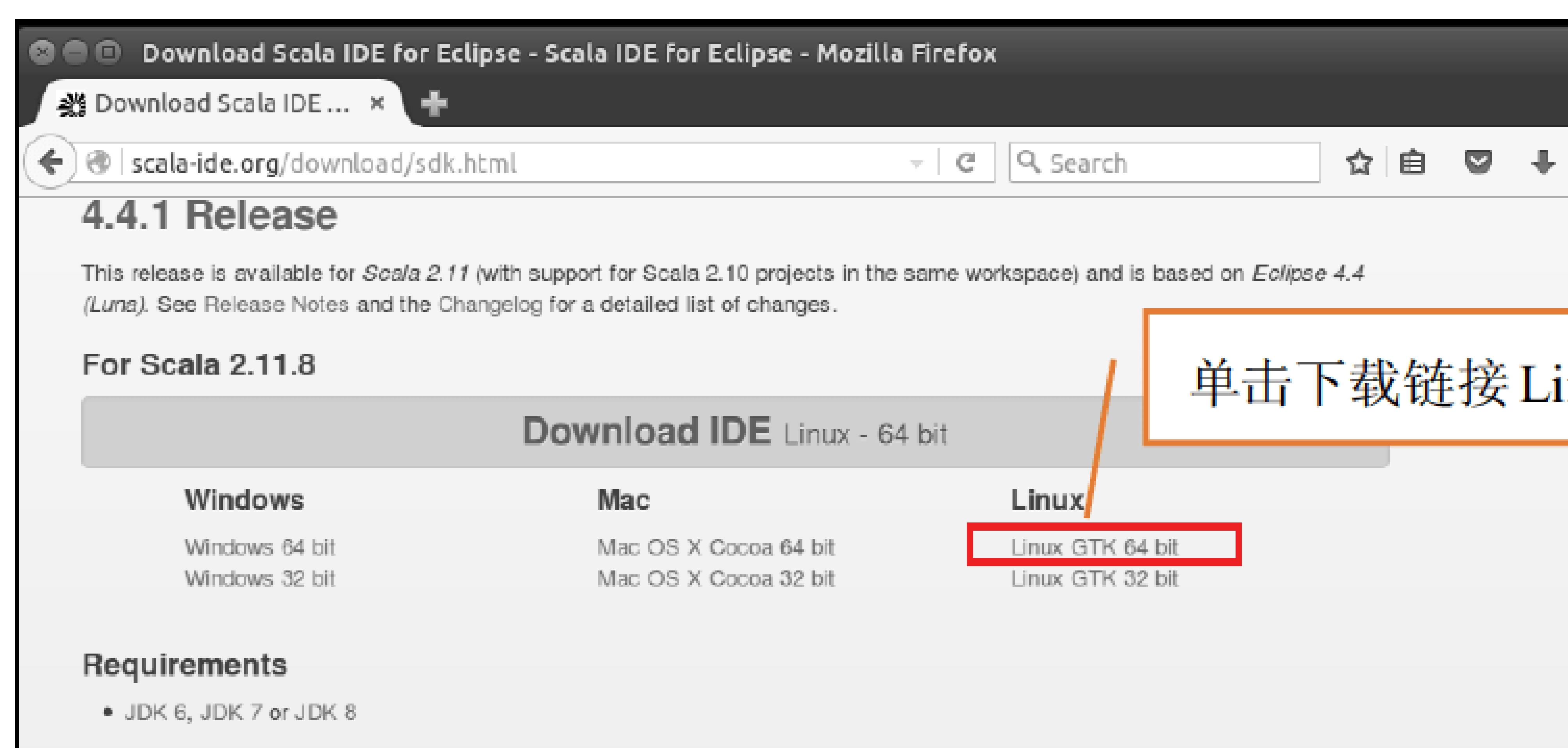


图 11-5 Scala IDE 下载页面

步骤03 选择以“归档管理器”来打开（见图 11-6）

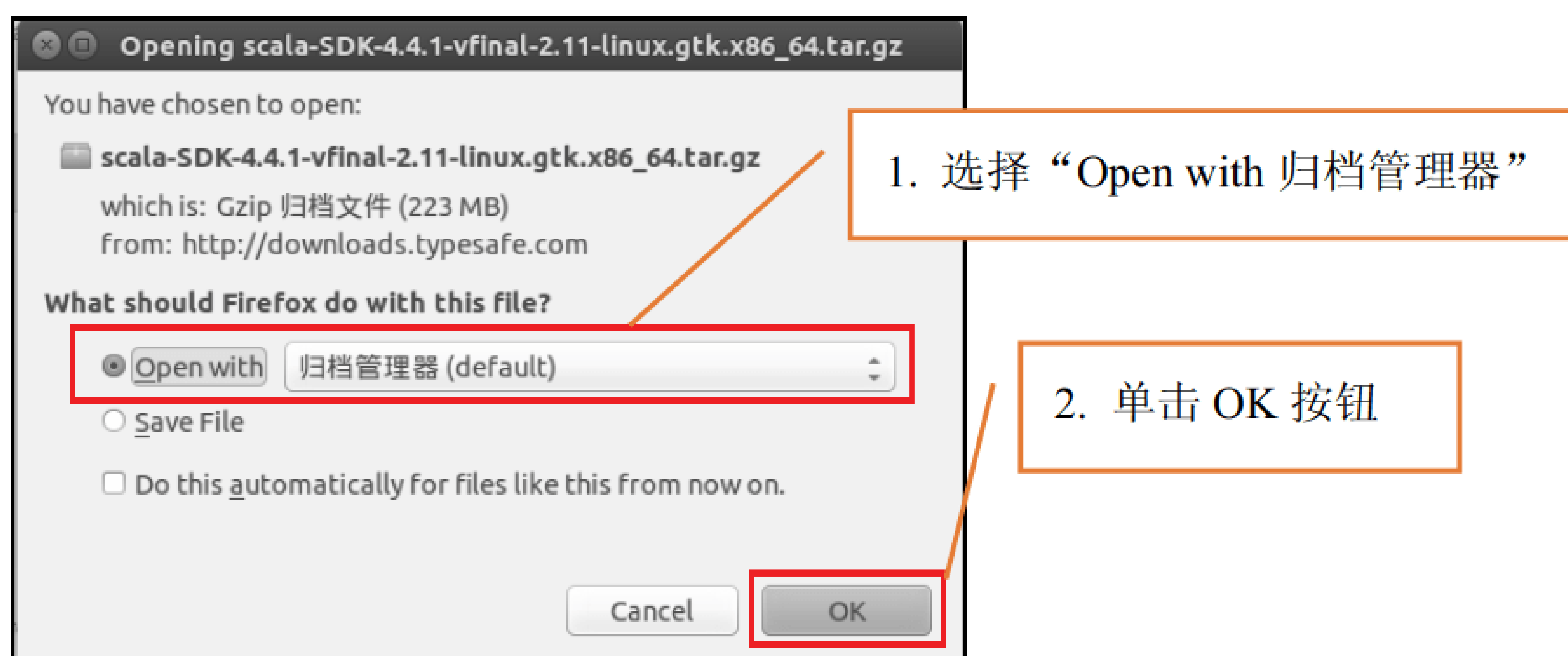


图 11-6　选择“Open with 归档管理器”

步骤 04 归档管理器界面

在归档管理器中解压缩文件，如图 11-7 所示。

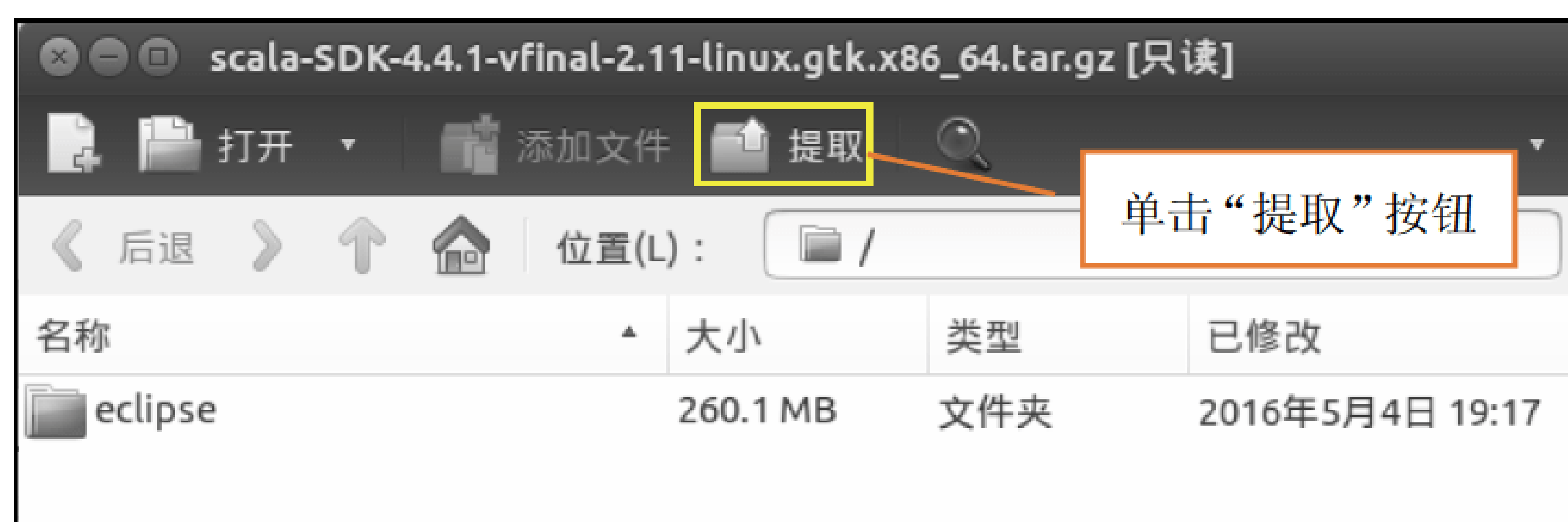


图 11-7　提取压缩文件

步骤 05 选择解压缩的文件夹

接下来选择解压缩的文件夹，解压缩到“主文件夹”，屏幕显示界面如图 11-8 所示。



图 11-8　选择解压缩的文件夹

步骤 06 启动文件资源管理器

启动文件资源管理器，然后打开 eclipse 文件夹，如图 11-9 所示。

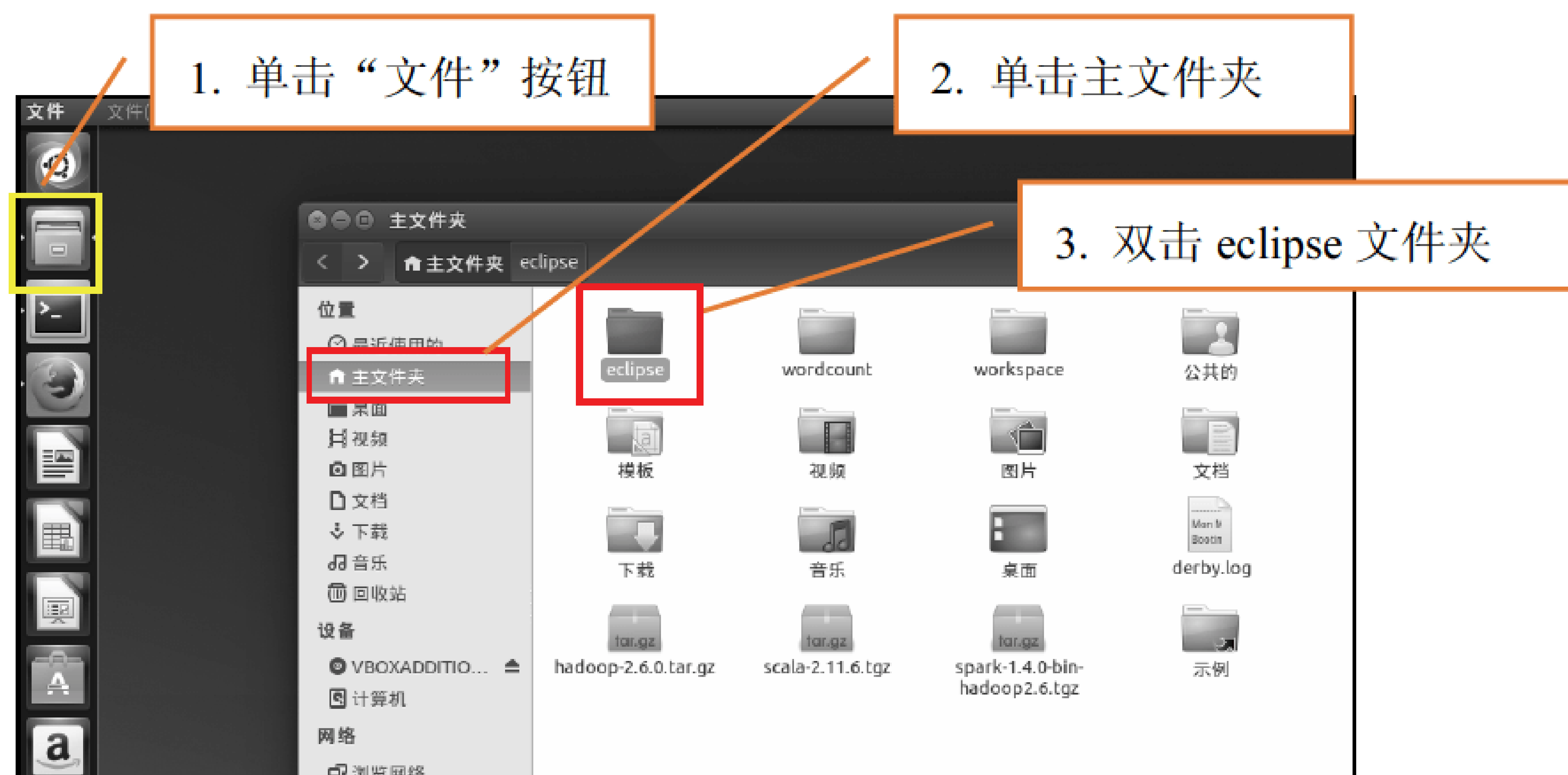


图 11-9　打开 eclipse 文件夹

步骤 07 创建链接

创建链接，方便以后运行程序，如图 11-10 所示。



图 11-10　创建链接

步骤 08 将创建的 eclipse 链接拖动到桌面

将创建的 eclipse 链接拖动到桌面。以后要运行 eclipse 时，只需要在图标上双击即可，如图 11-11 所示。

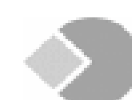




图 11-11　拖动链接至桌面

11.2　安装 PyDev

开发 Python 使用 eclipse 作为 IDE，必须安装 PyDev 这个 eclipse 的 plugin。

步骤 01　运行 eclipse 程序

输入工作路径/home/hduser/pythonwork，步骤如图 11-12 所示。

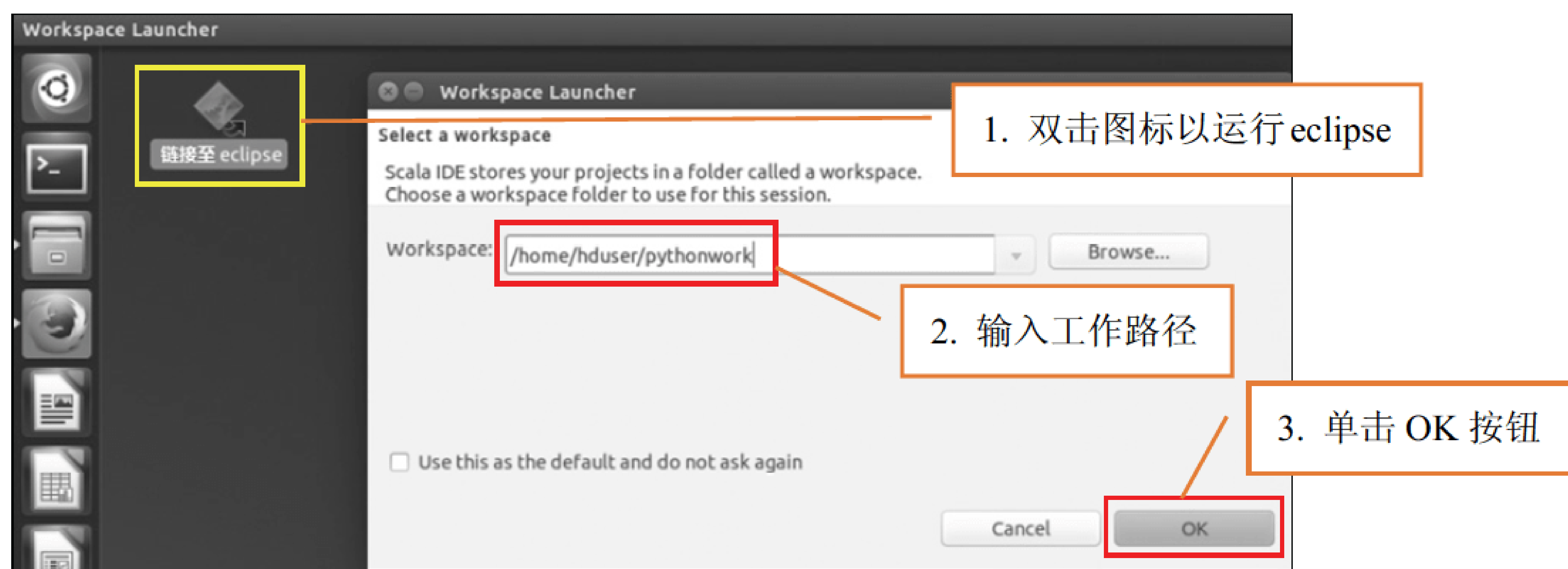


图 11-12　运行 eclipse 程序

步骤 02　安装 PyDev 程序包

启动后的 eclipse 屏幕显示界面如图 11-13 所示，并按照下列步骤安装 PyDev 程序包。

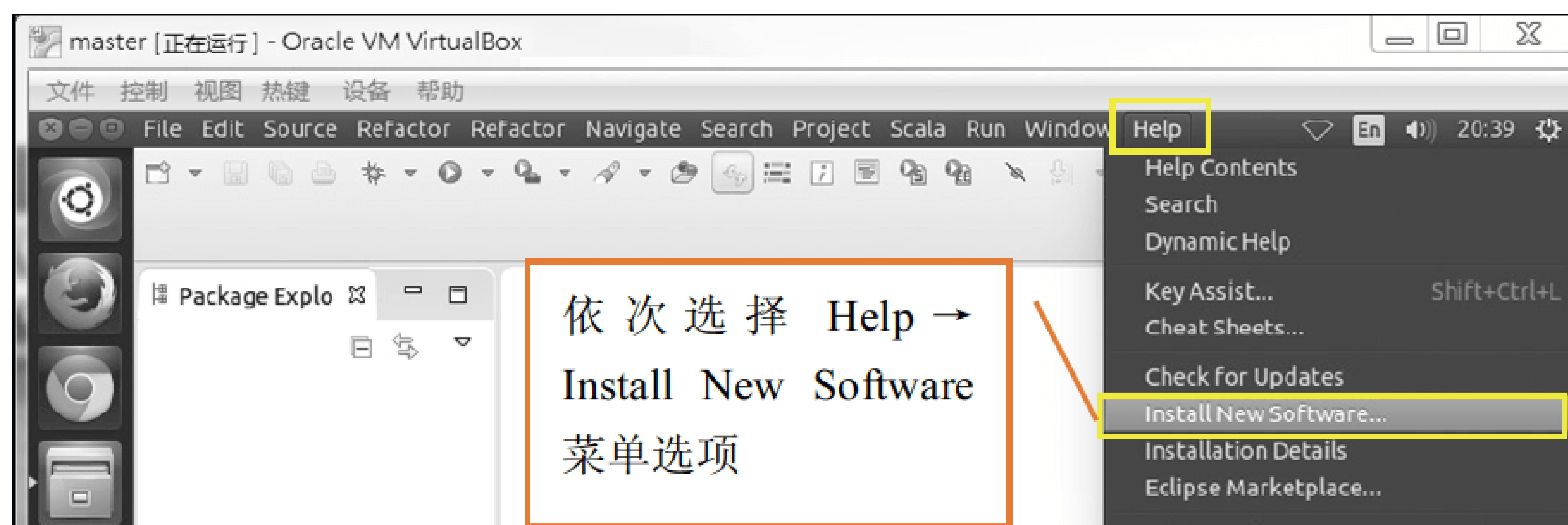


图 11-13 选择菜单选项

步骤 03 添加程序包

选择菜单选项后会打开 Install 对话框，单击 Add 按钮，之后会打开 Add Repository 对话框。输入要安装程序包的网址，eclipse 会帮助我们下载安装 PyDev 4.5.4 版本，如图 11-14 所示。

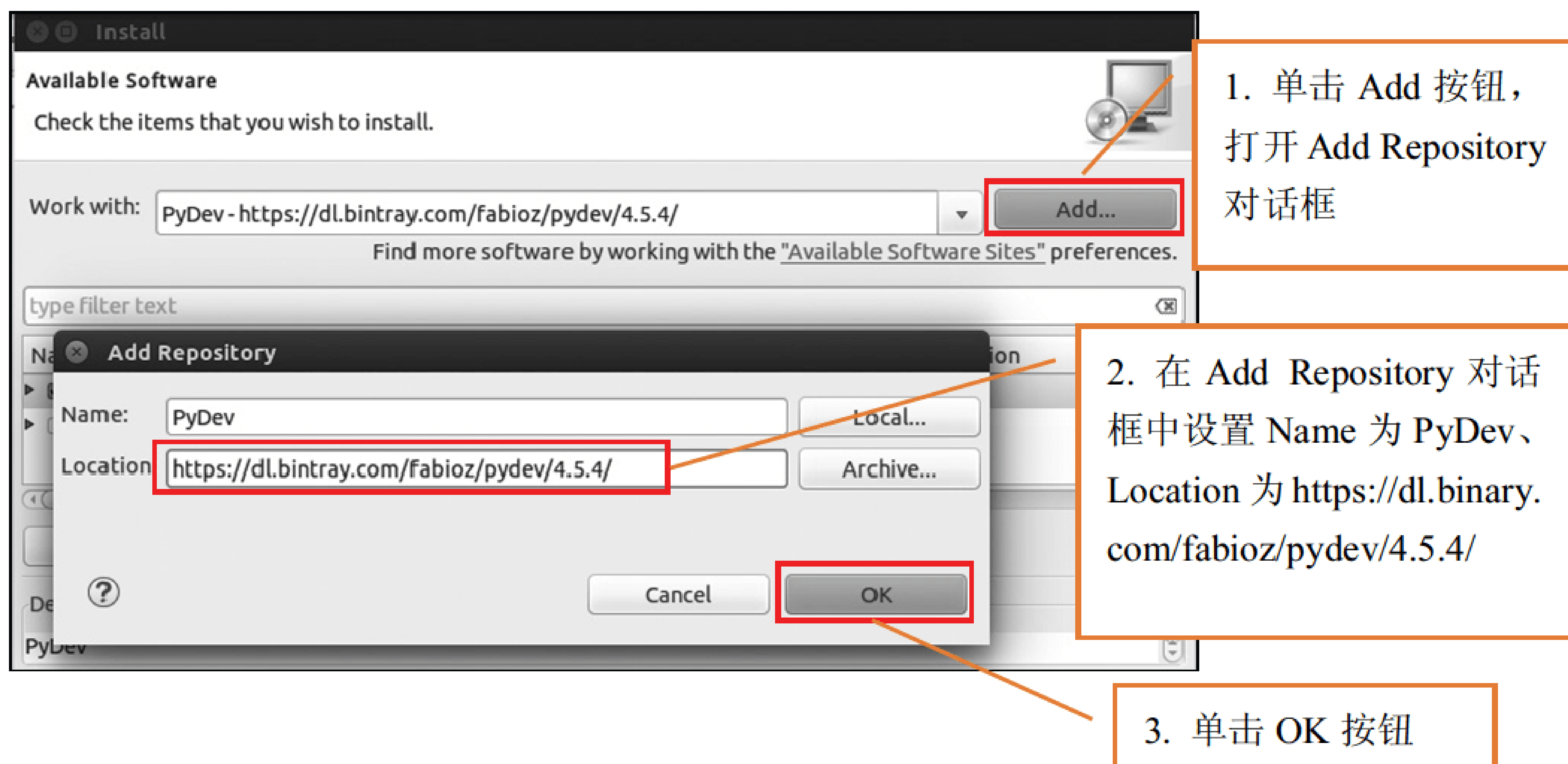


图 11-14 添加程序包

步骤 04 Add Repository 对话框

等候一些时间后，屏幕显示界面会出现 PyDev 安装选项，先勾选 PyDev，然后单击 Next 按钮，如图 11-15 所示。



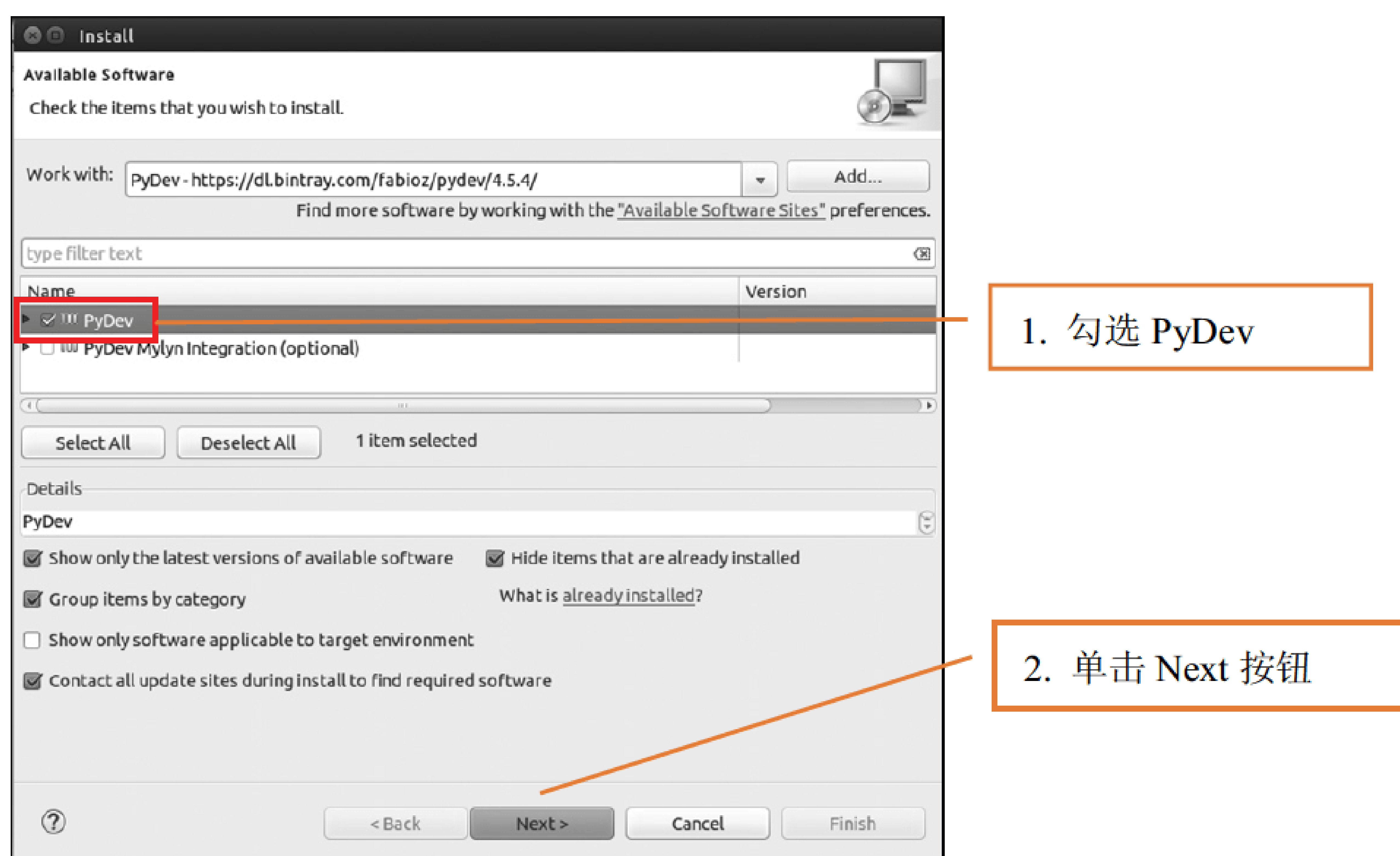


图 11-15　选择安装 PyDev

步骤 05　显示安装选项（见图 11-16）

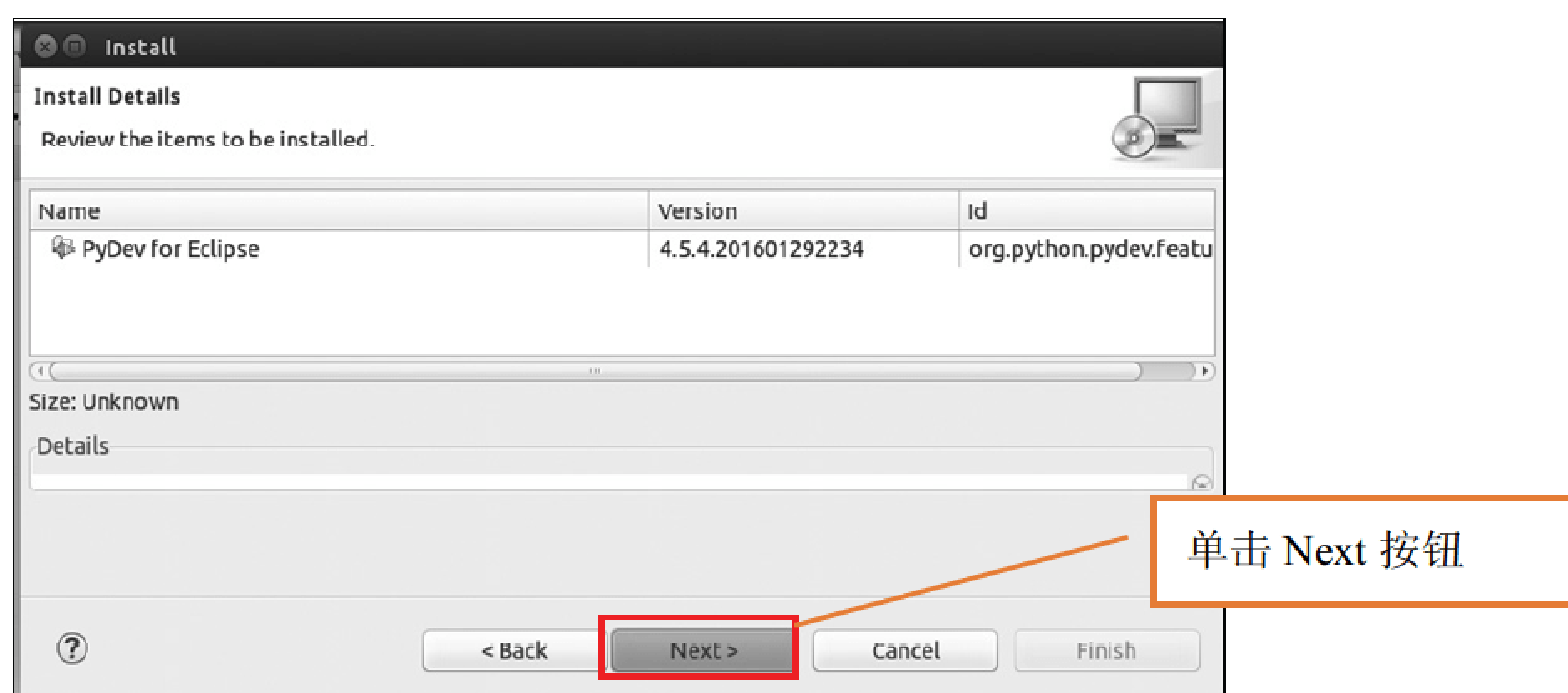


图 11-16　显示安装选项

步骤 06　阅读Licenses 条款（见图 11-17）

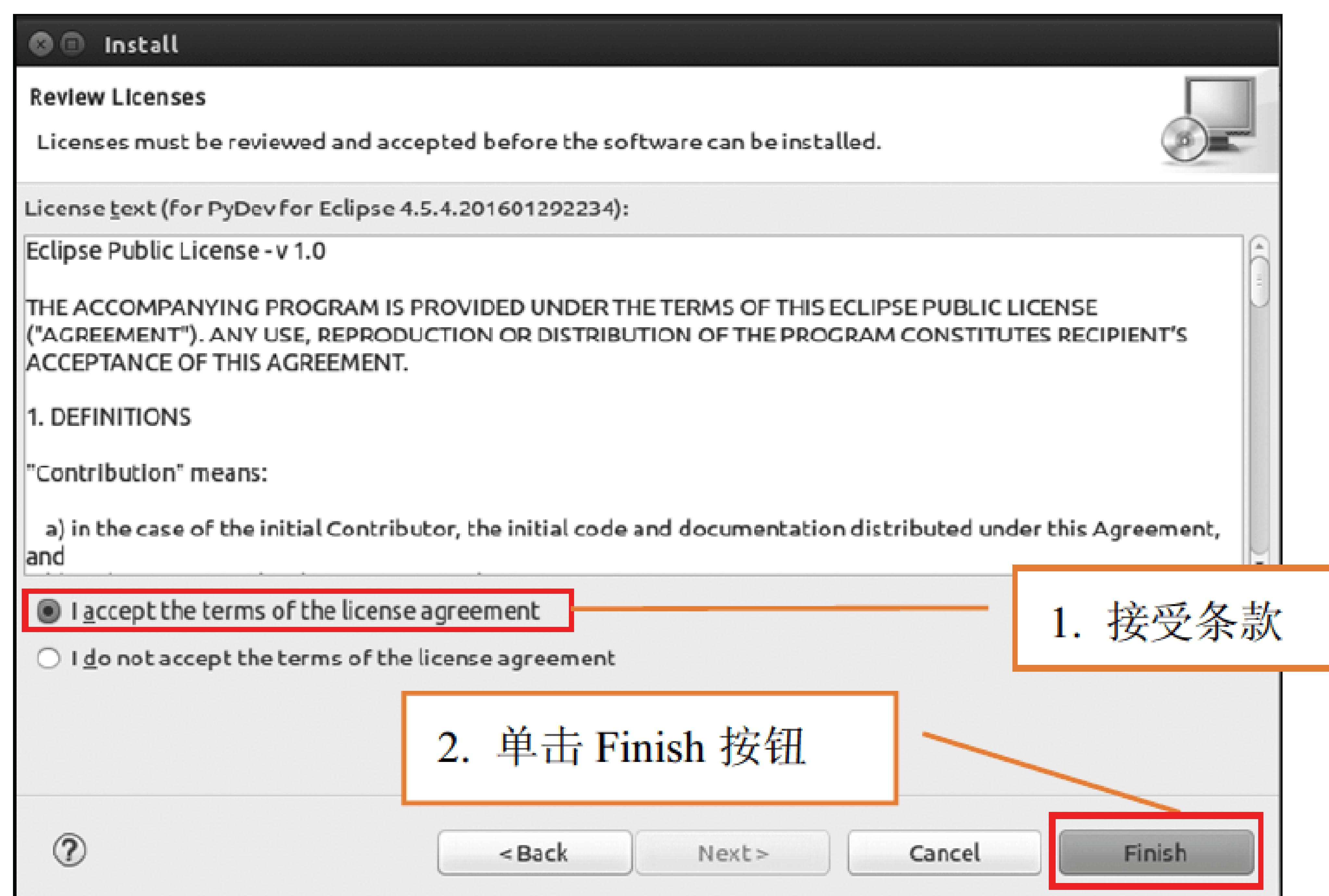


图 11-17　阅读 Licenses 条款并接受

步骤 07　安装进度（见图 11-18）

设置完之前的步骤后单击 Finish 按钮，就会回到 eclipse 主界面。此时将会开始安装 PyDev。我们可以在 eclipse 界面的右下角看到安装的进度，等候安装进度至 100%。

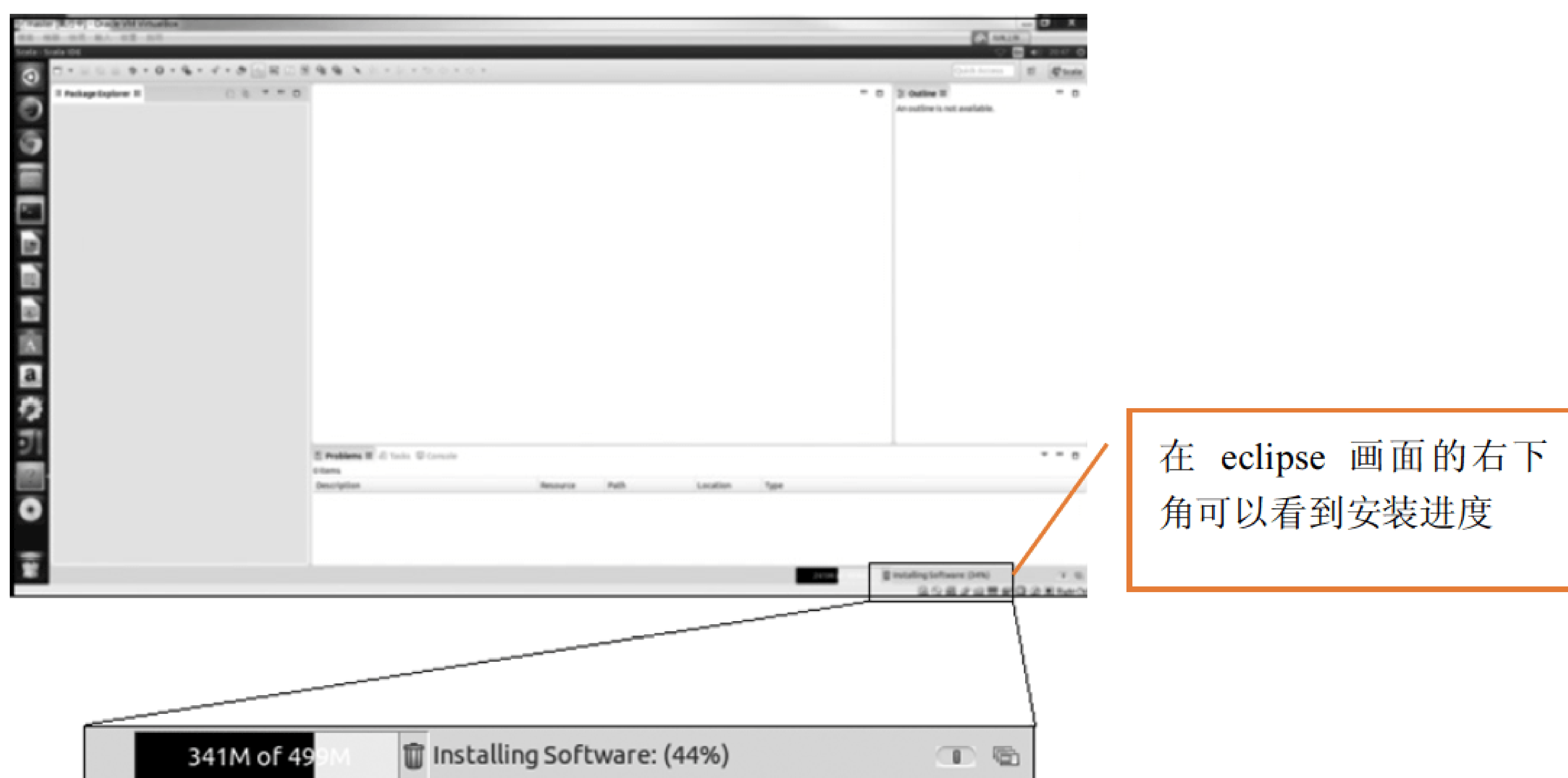


图 11-18　显示安装进度

步骤 08　确认是否信任下列认证

安装期间会询问是否信任认证（见图 11-19）。



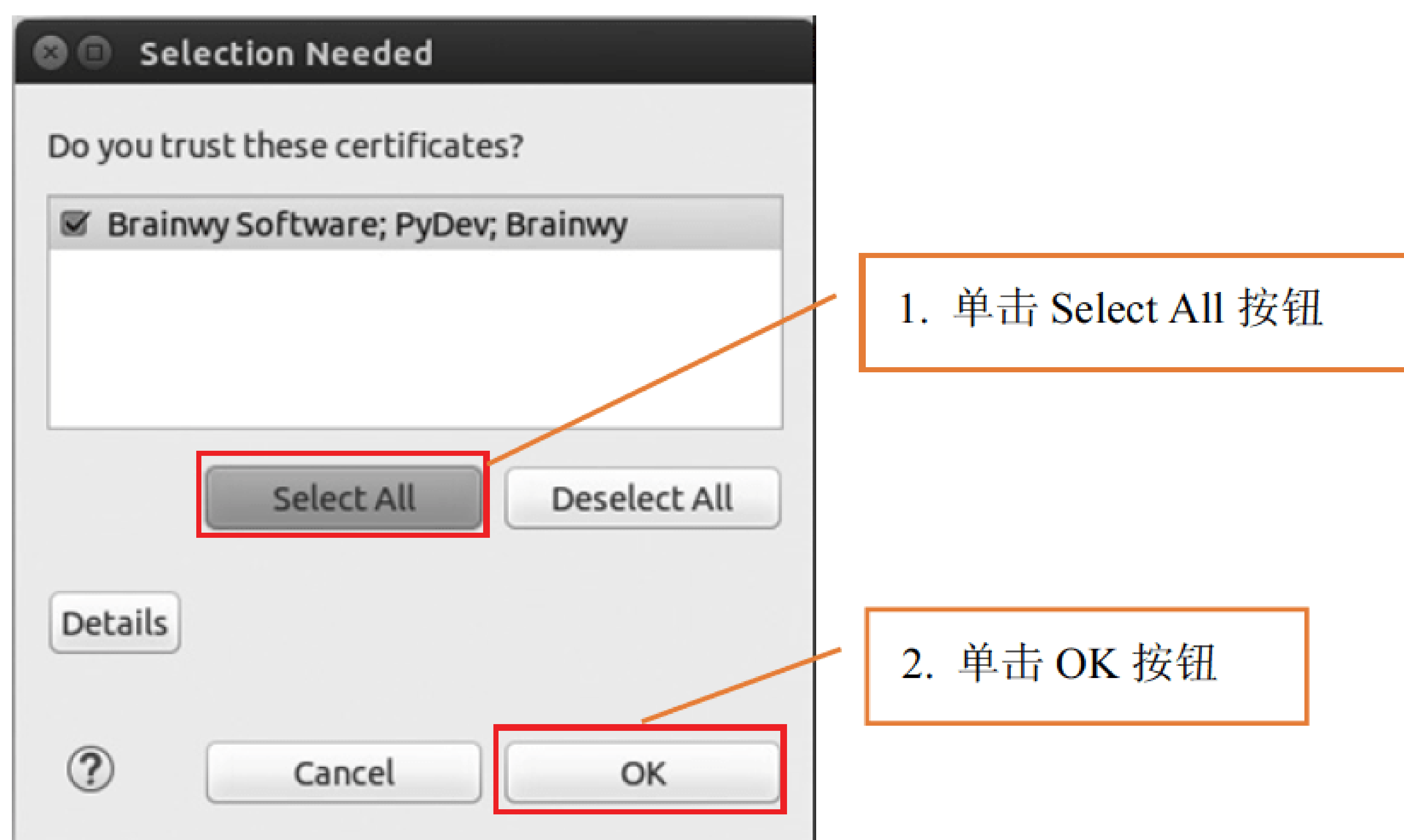


图 11-19 选择信任认证

步骤 09 询问是否要重新启动 eclipse

安装完成后，eclipse 必须重新启动才能生效（见图 11-20）。

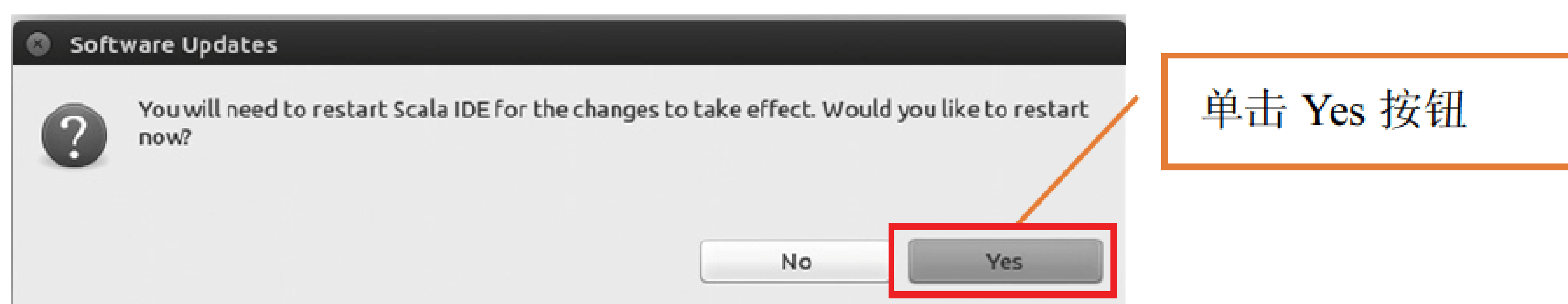


图 11-20 选择重新启动

步骤 10 重新启动 eclipse

重新启动后，输入工作路径（见图 11-21）。

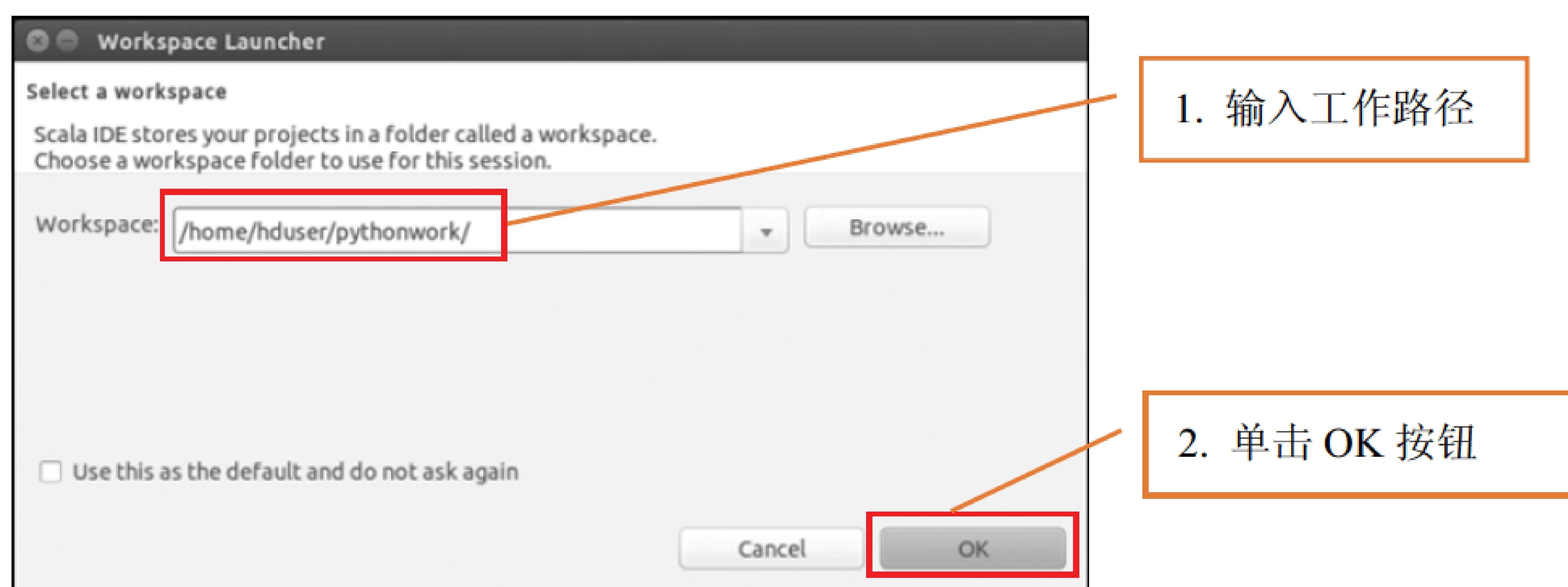


图 11-21 重新启动后输入工作路径

步骤 11 询问是否更新

eclipse 会询问是否更新，单击 Apply selected changes 按钮，如图 11-22 所示。

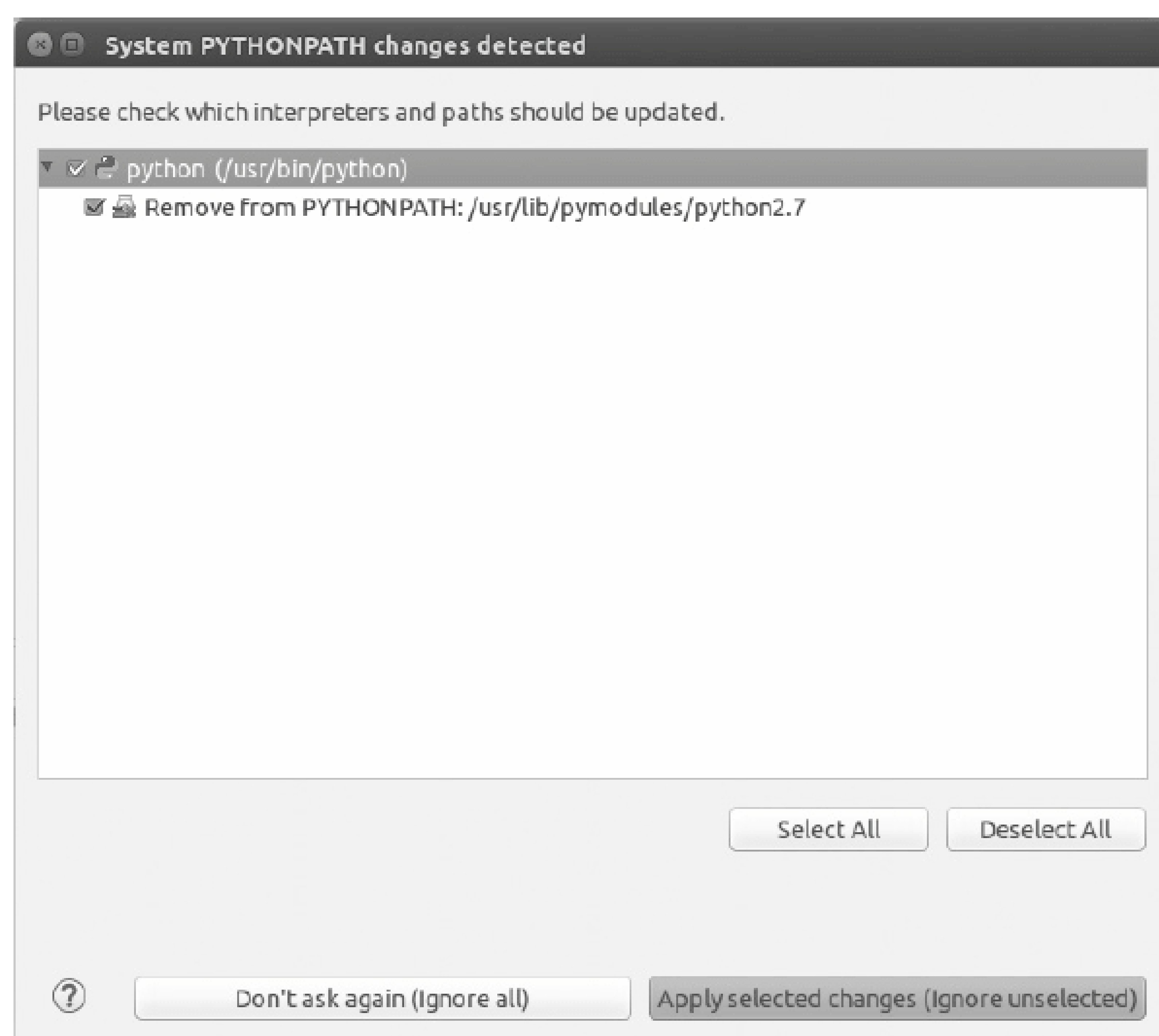


图 11-22　接受更新

11.3 设置字符串替代变量

字符串替代变量（String Substitution Variable）的作用是可以设置某一个变量来代表某一个字符串。例如，后续我们要设置 ${SPARK_HOME} 字符串替代变量为 Spark 安装路径。我们将设置 3 个字符串替代变量，这些替代变量在后续的设置过程都会使用到。

- SPARK_HOME= /usr/local/spark（Spark 的安装路径）。
- HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop（Hadoop 配置文件的路径）。
- PYSPARK_PYTHON= /home/hduser/anaconda2/bin/python（anaconda 链接库路径）。

步骤 01 依次选择 Window→Preferences 菜单选项（见图 11-23）

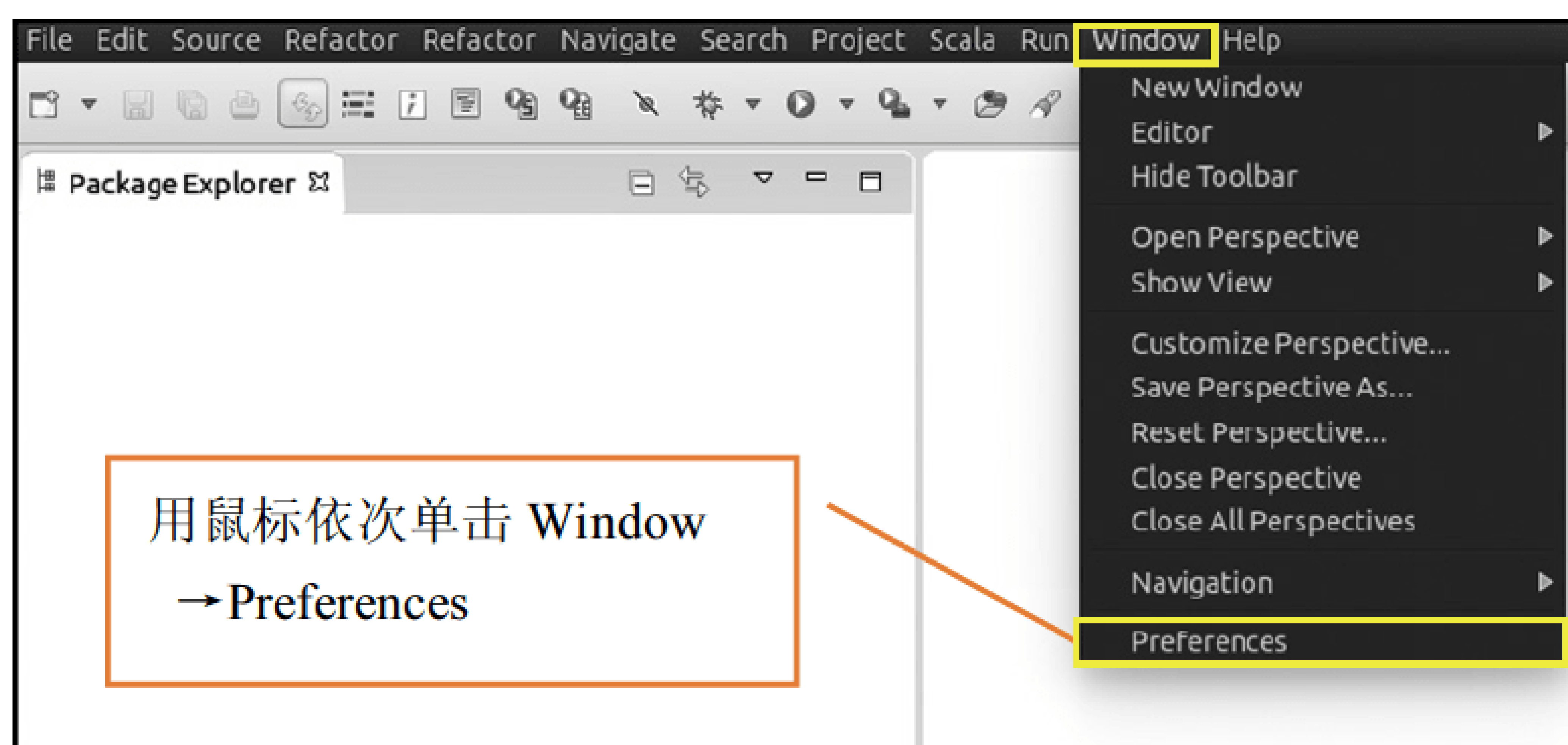


图 11-23　选择菜单选项



步骤 02 设置字符串替代变量

依次单击 Run/Debug → String Substitution 菜单选项来设置字符串替代，如图 11-24 所示。

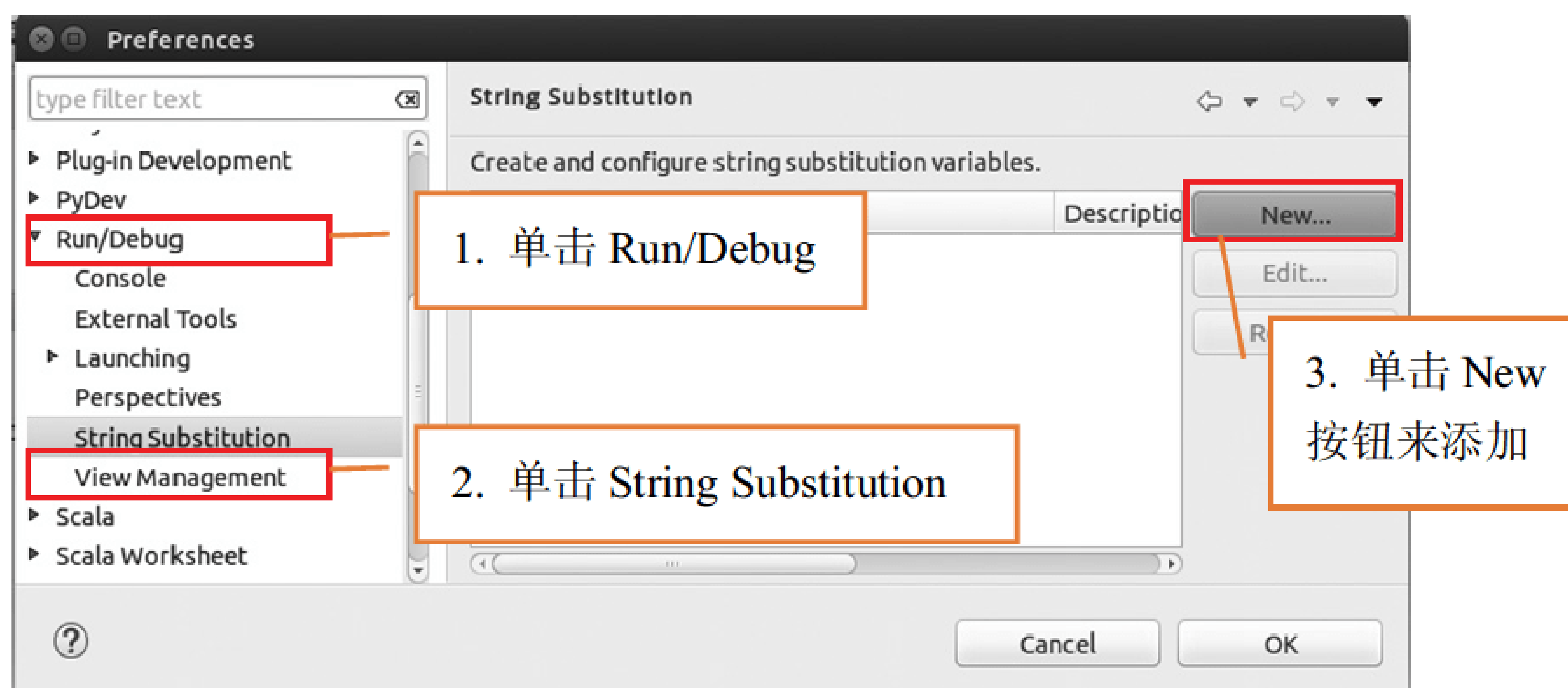


图 11-24　设置字符串替代变量

步骤 03 设置 SPARK_HOME 字符串替代变量（见图 11-25）

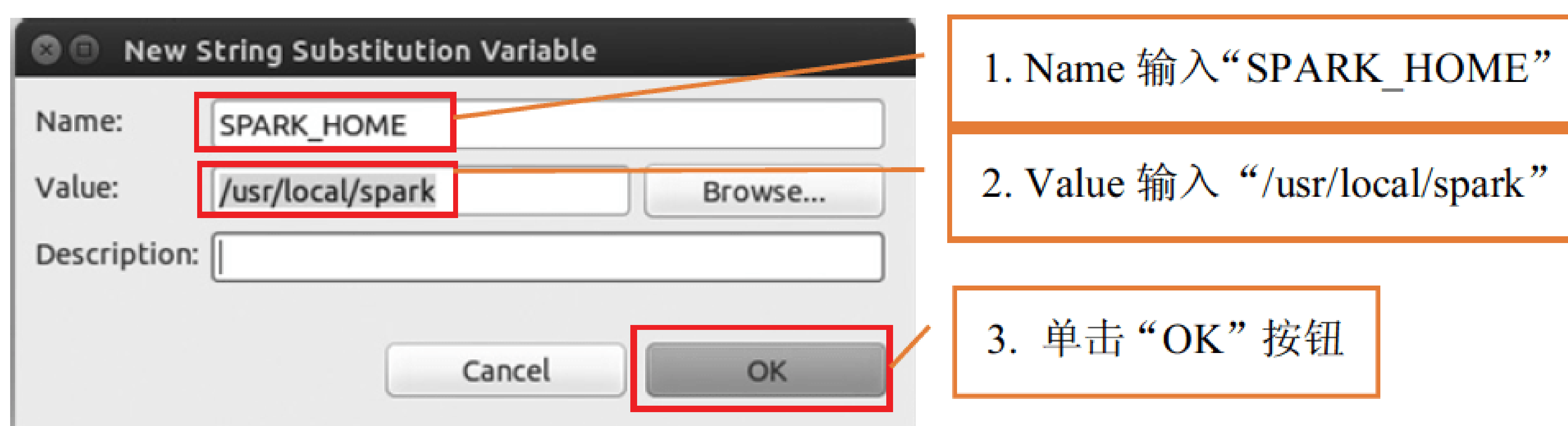


图 11-25　设置 SPARK_HOME 字符串替代变量

步骤 04 确认已设置 SPARK_HOME 字符串替代变量

回到设置字符串替代变量的屏幕显示界面，确认 SPARK_HOME 已设置好了。然后单击 New 按钮添加其他字符串替代变量，如图 11-26 所示。



图 11-26　确认已设置的 SPARK_HOME 字符串替代变量

步骤05 设置 HADOOP_CONF_DIR 字符串替代变量（见图 11-27）

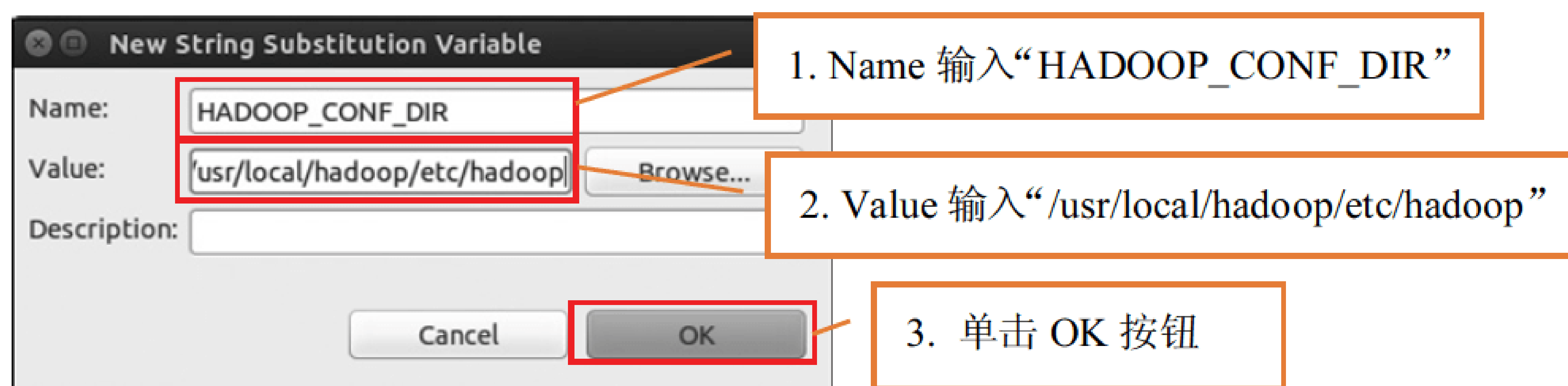


图 11-27 设置 HADOOP_CONF_DIR 字符串替代变量

步骤06 确认已设置 HADOOP_CONF_DIR 字符串替代变量

回到设置字符串替代变量界面，确认 HADOOP_CONF_DIR 已设置好了。然后单击 New 按钮添加其他字符串替代变量，如图 11-28 所示。



图 11-28 确认已设置 HADOOP_CONF_DIR 字符串替代变量

步骤07 设置 PYSPARK_PYTHON 字符串替代变量（见图 11-29）

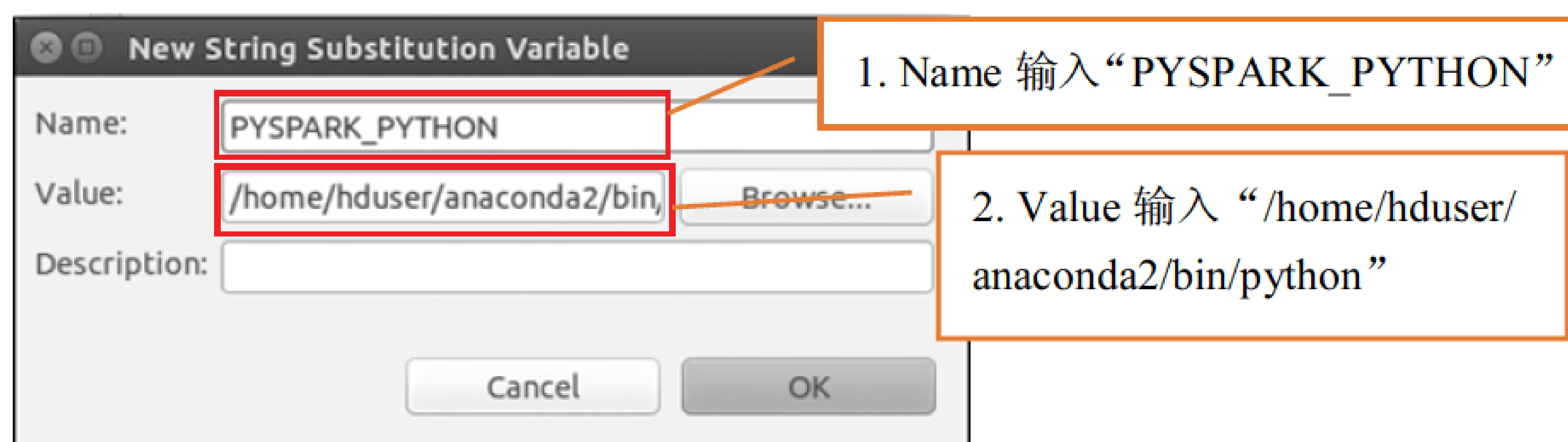


图 11-29 设置 PYSPARK_PYTHON 字符串替代变量

步骤08 确认已设置 PYSPARK_PYTHON 字符串替代变量

回到设置字符串替代变量的屏幕显示界面，确认到 PYSPARK_PYTHON 已设置，如图 11-30 所示。



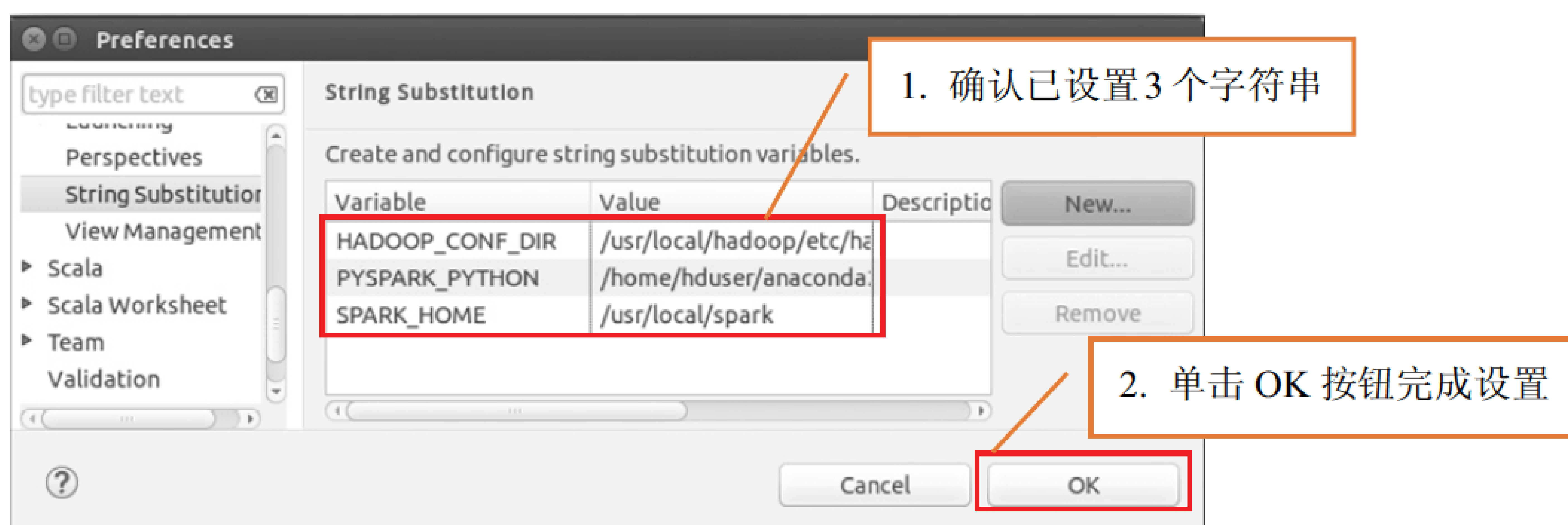


图 11-30　确认已设置 3 个字符串替代变量

11.4　PyDev 设置 Python 链接库

Python 是解释性语言，所以必须设置解释器（Interpreter）。

步骤 01 依次选择 Window → Preferences 菜单选项（见图 11-31）

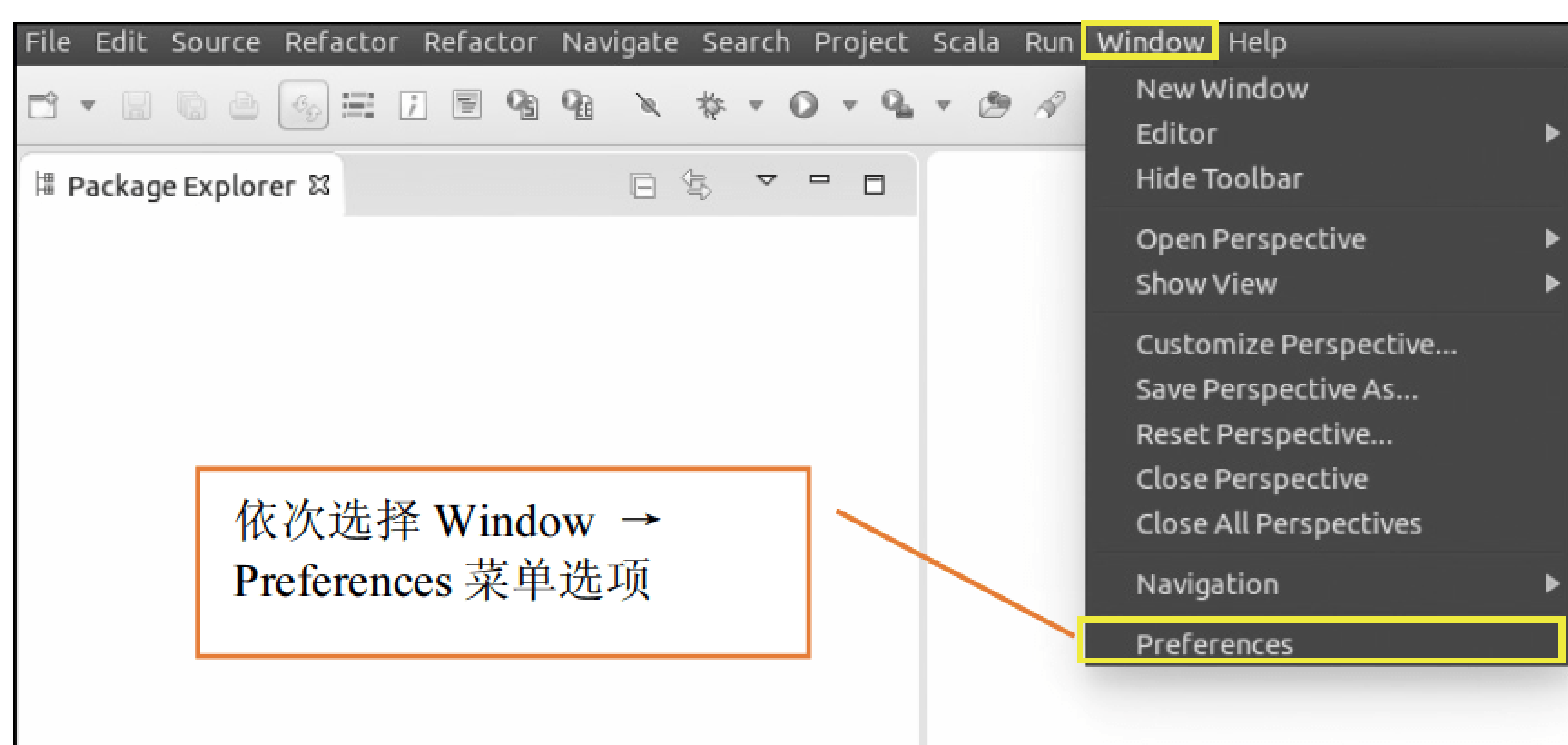


图 11-31　选择菜单选项

步骤 02 PyDev→Interpreter→Python Interpreter（见图 11-32）



图 11-32 打开 Python Interpreter 设置界面

步骤 03 自动配置设置——搜索适合安装的版本

单击 Advanced Auto-Config，系统会搜索适合安装的 Python 版本让我们选择（如图 11-33 所示）。

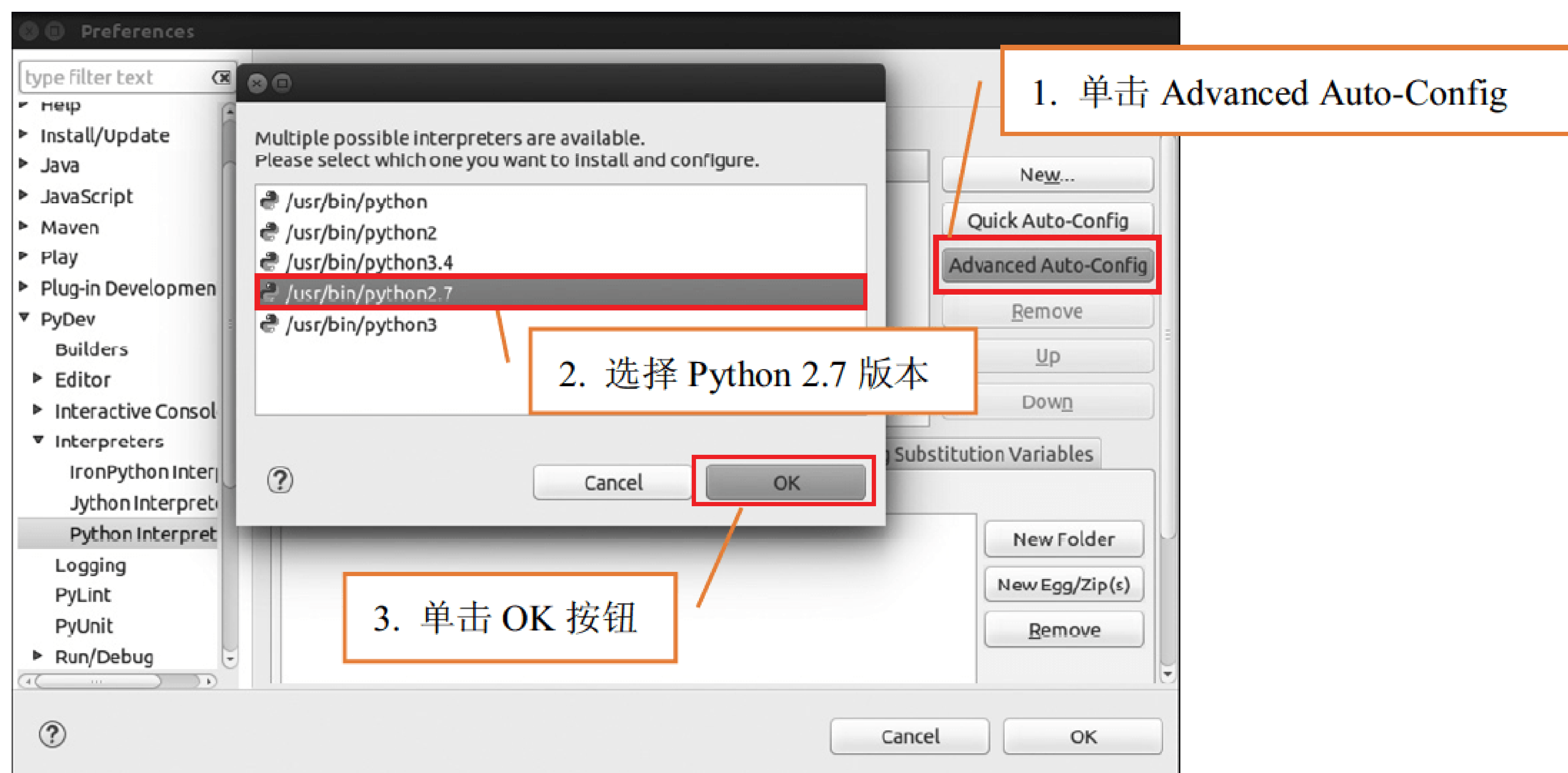


图 11-33 搜索适合安装的版本

步骤 04 自动配置设置——显示需要安装的链接库

系统会显示需要安装的 lib 链接库，单击 OK 按钮，如图 11-34 所示。



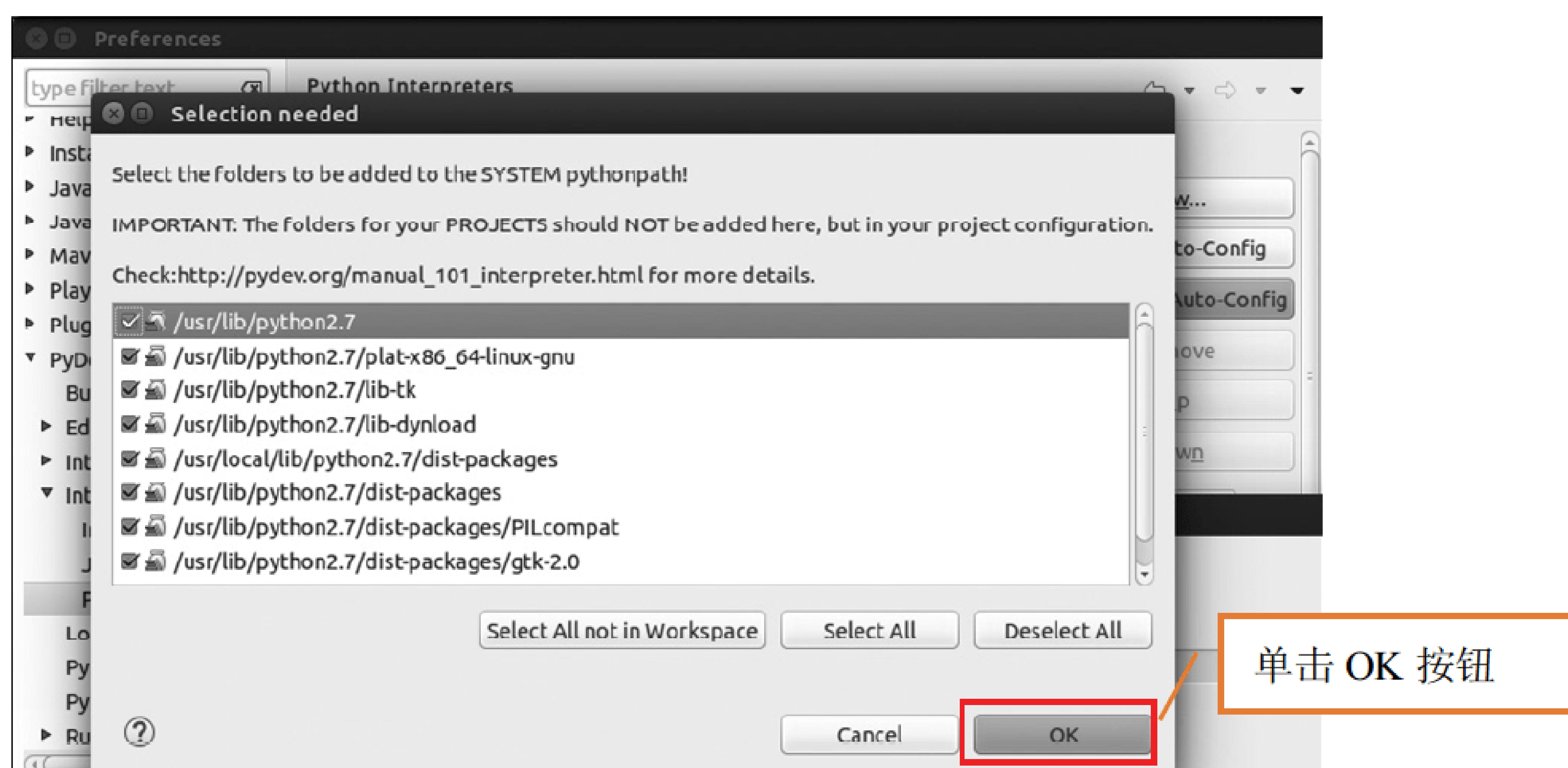


图 11-34　显示需要安装的链接库

步骤 05 确认已安装的 lib 链接库（见图 11-35）

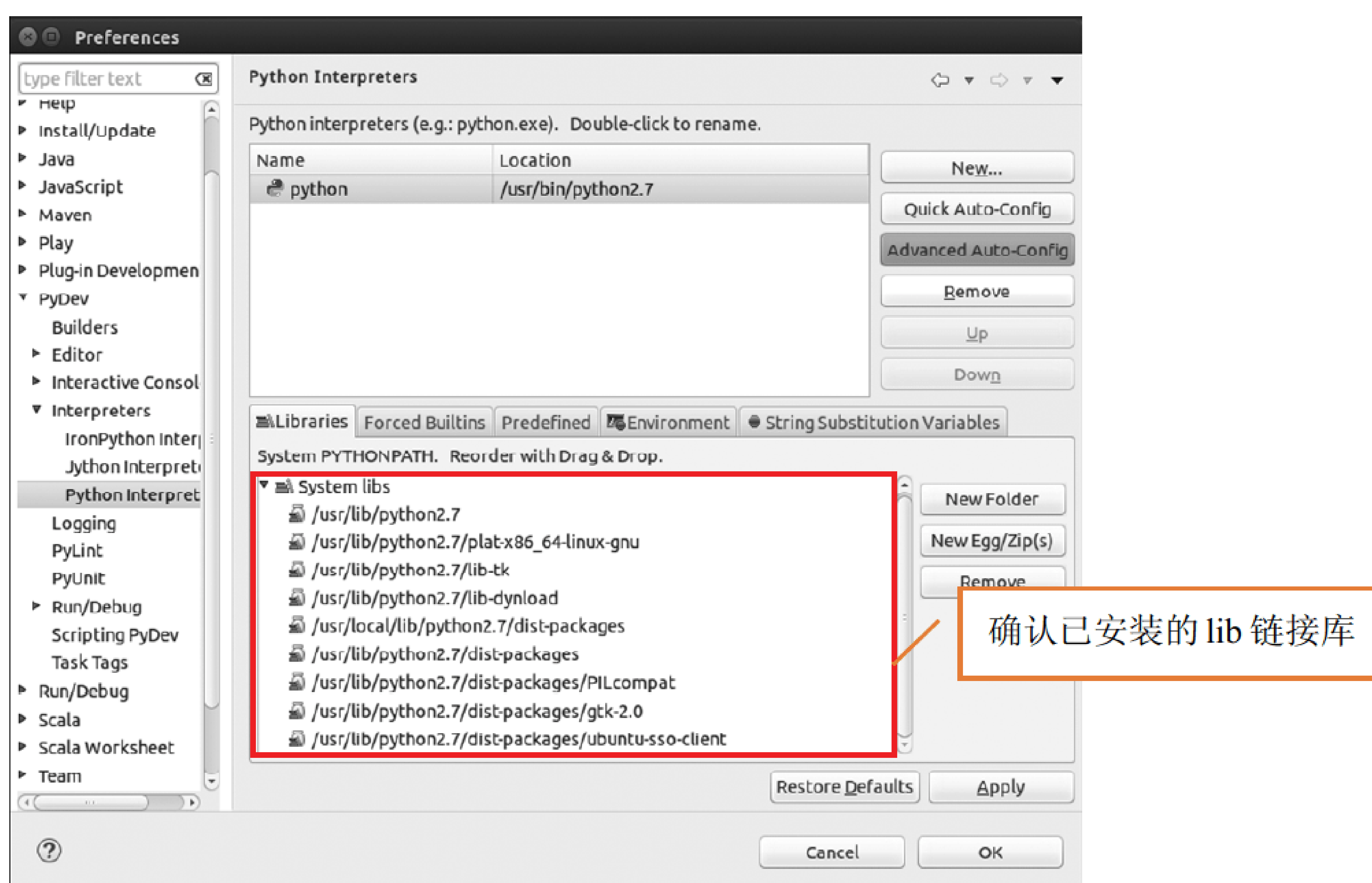


图 11-35　确认已安装的链接库

11.5 PyDev 设置 anaconda2 链接库路径

后续我们会使用到一些 python 链接库，例如 numpy、pandas 所以我们必须添加 anaconda2 路径 /home/hduser/anaconda2/lib/python2.7/site-packages。

步骤 01 单击 New Folder 按钮（见图 11-36）

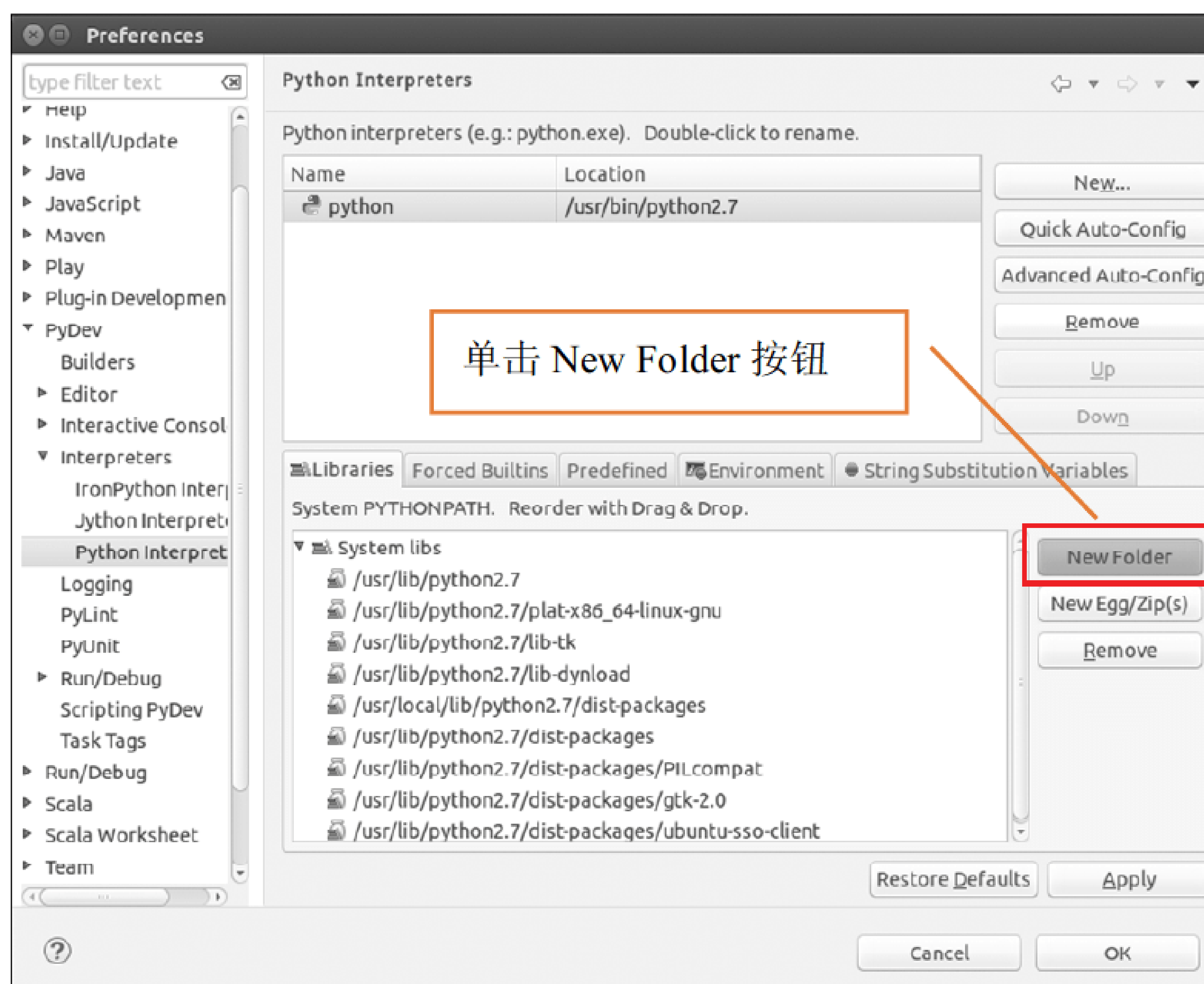


图 11-36　单击 New Folder 按钮

步骤 02 设置 /home/hduser/anaconda2/lib/python2.7/site-packages 路径（见图 11-37）

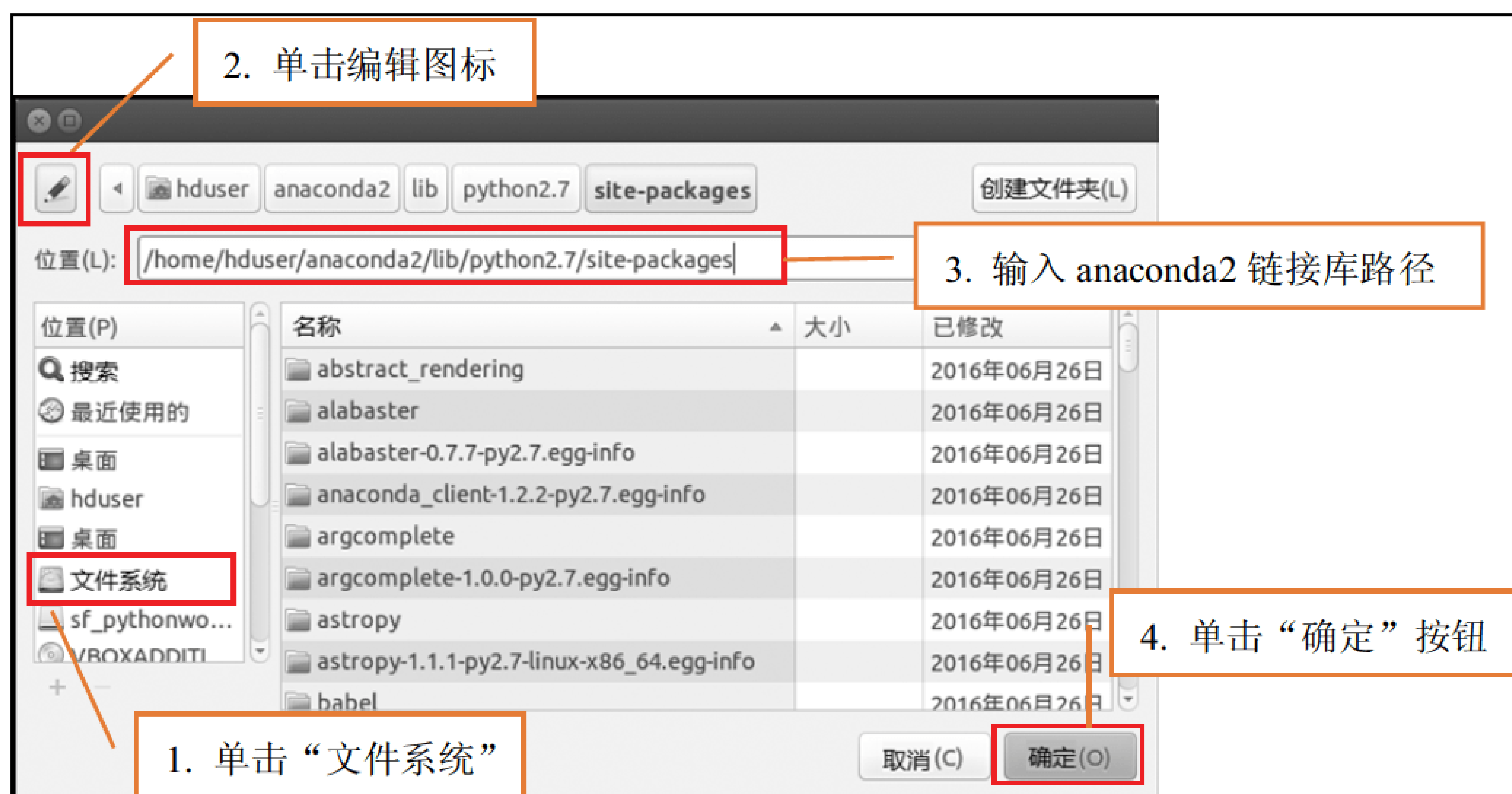


图 11-37　设置 anaconda2 链接库路径

步骤 03 确认已经设置路径

确认已经添加 hduser/anaconda2/lib/python2.7/site-packages 路径，如图 11-38 所示。



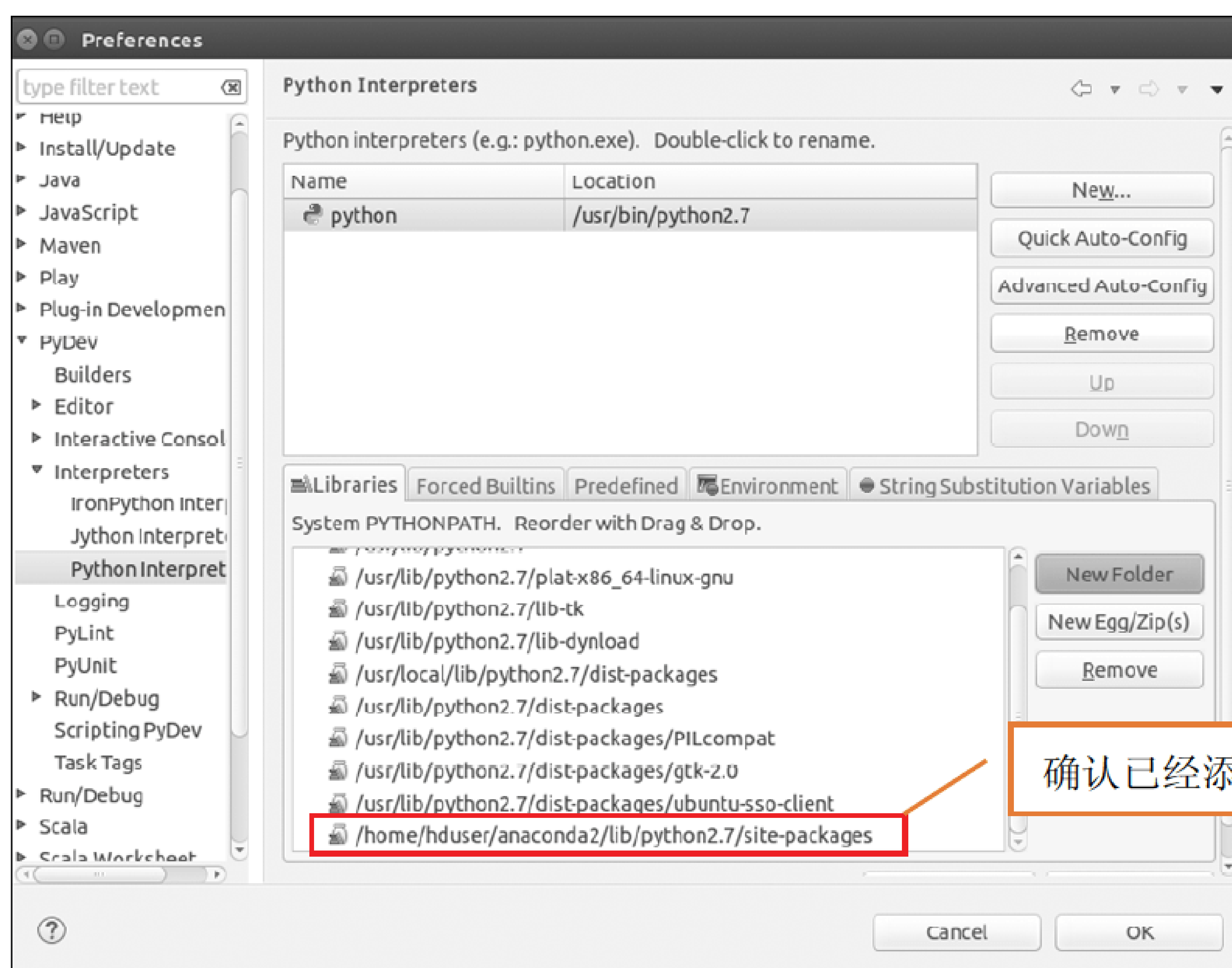


图 11-38　确认已经设置路径

11.6 PyDev 设置 Spark Python 链接库

当我们执行 Spark Python 程序时，必须引用 Spark Python 链接库，所以我们必须设置 PyDev 解释器 Spark Python 链接库才能够正确执行。

步骤 01　设置 Spark Python 链接库（见图 11-39）

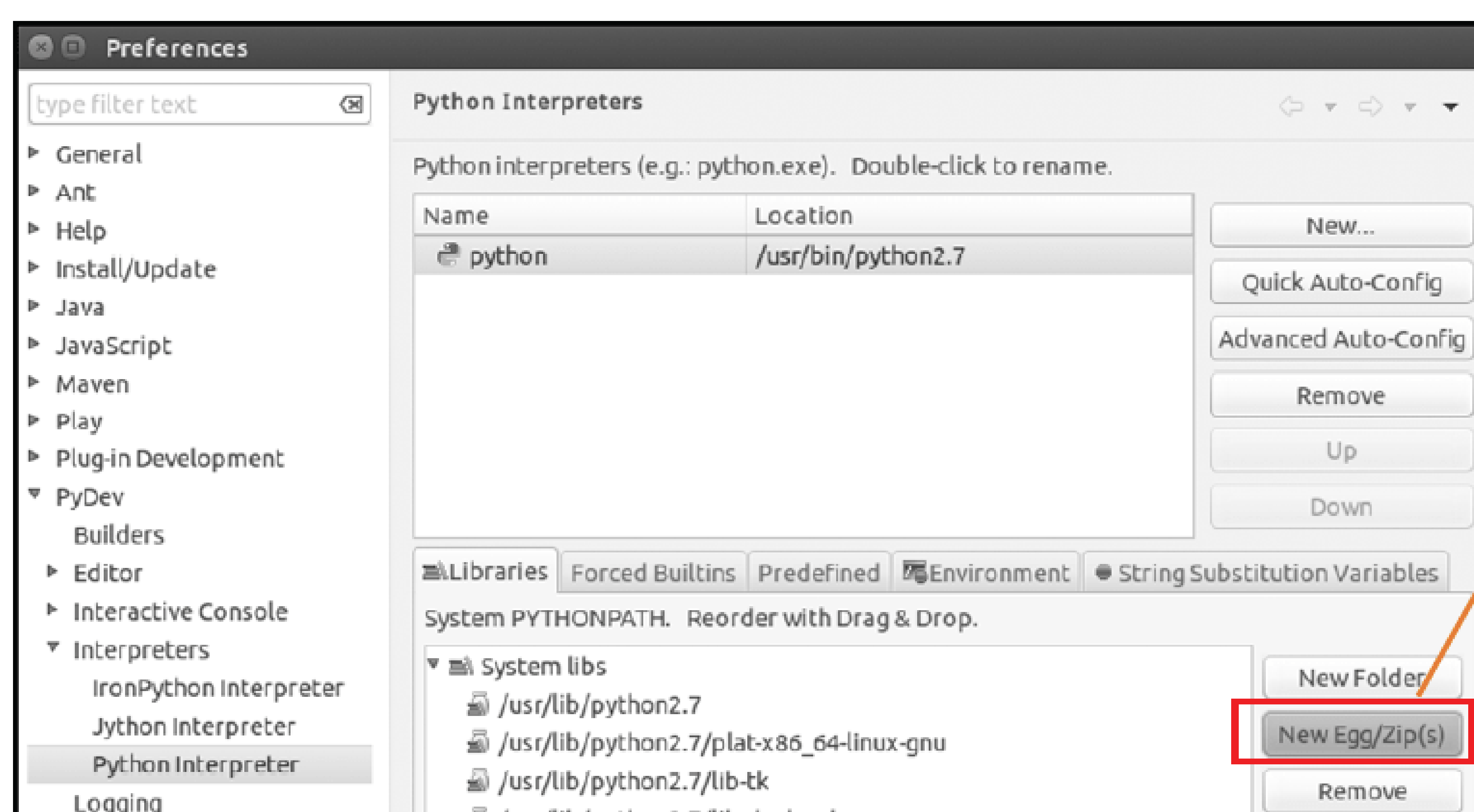


图 11-39　设置 Spark Python 链接库

步骤02 选择链接库

Spark 的 Python 链接库存储在/usr/local/spark/python/lib 路径中，有两个文件：py4j-0.10.1-src.zip 和 pyspark.zip，如图 11-40 所示。因为链接库的扩展名是 zip，所以必须选择*.zip 的链接库。



图 11-40 选择链接库

步骤03 确认已经安装的链接库（见图 11-41）

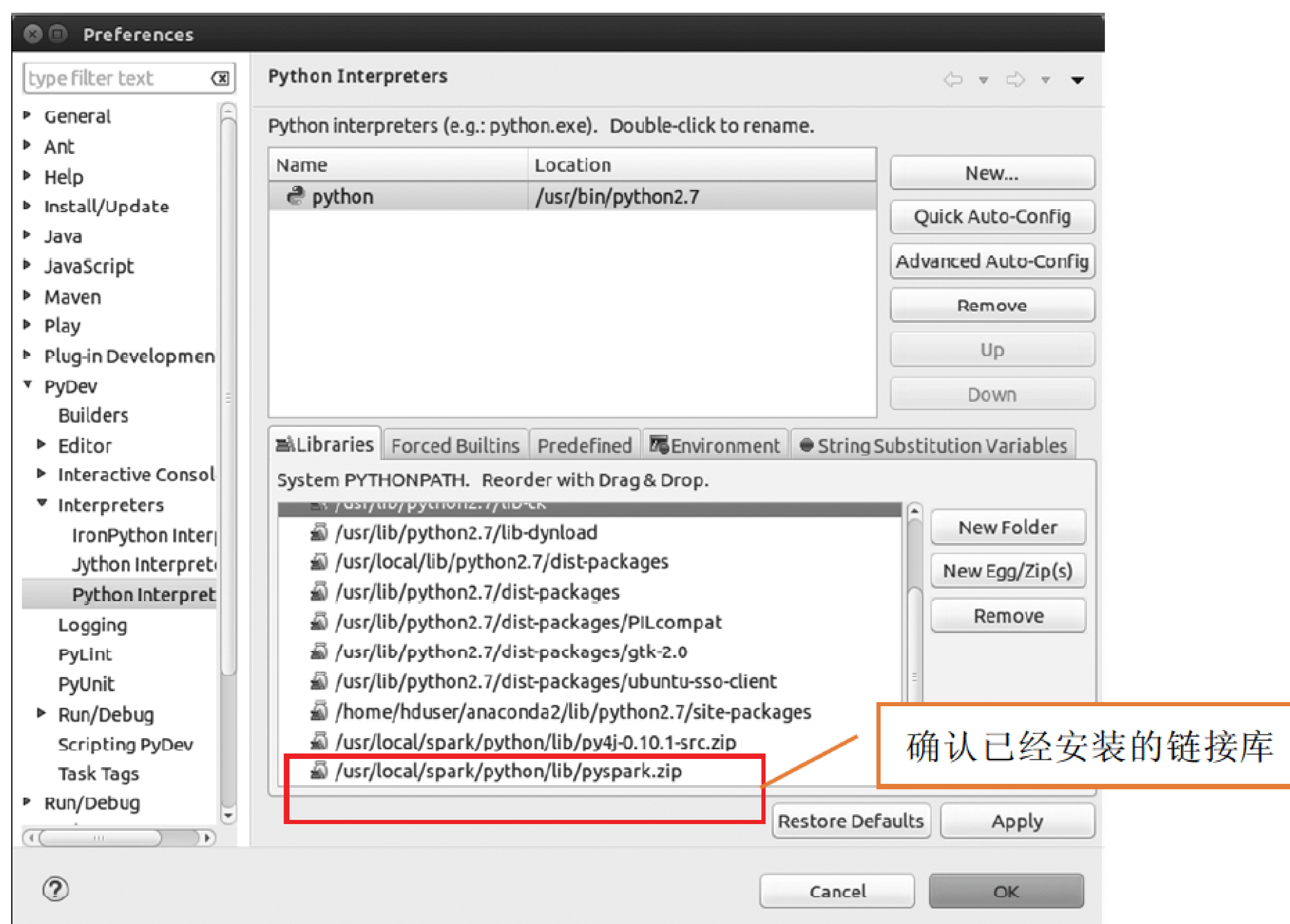


图 11-41 确认已经安装的链接库

11.7 PyDev 设置环境变量

当我们在 eclipse 执行 Spark Python 程序时，必须设置以下两个环境变量：



- 当我们在 eclipse 执行 Spark Python 程序时，必须设置 SPARK_HOME（Spark 的安装路径）环境变量。
- 当我们在 eclipse 执行 Python Spark 程序时，必须读取 HDFS 文件，所以必须设置 HADOOP_CONF_DIR 环境变量（这是Hadoop 配置文件的路径）。

步骤01 切换到 Environment 页面（见图 11-42）

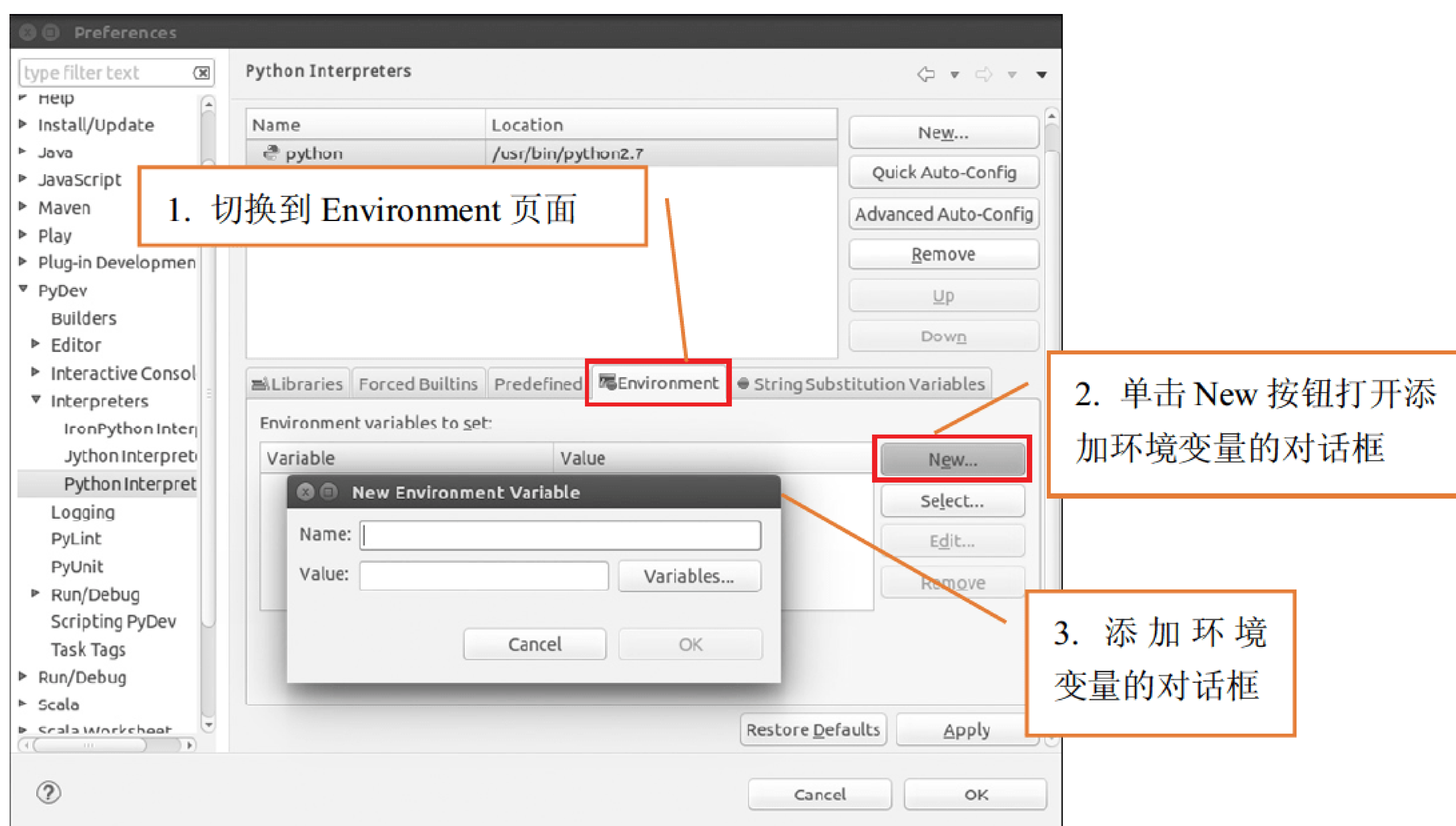


图 11-42　切换到 Environment 页面进行设置

步骤02 添加 SPARK_HOME 环境变量

在添加环境变量的对话框中，将 Name 设置为 SPARK_HOME、Value 设置为 ${SPARK_HOME}，如图 11-43 所示。其中，${SPARK_HOME}是代表我们在第 11.3 节设置的字符串替代变量。



图 11-43　添加 SPARK_HOME 环境变量

步骤03 确认已创建 SPARK_HOME 环境变量（见图 11-44）

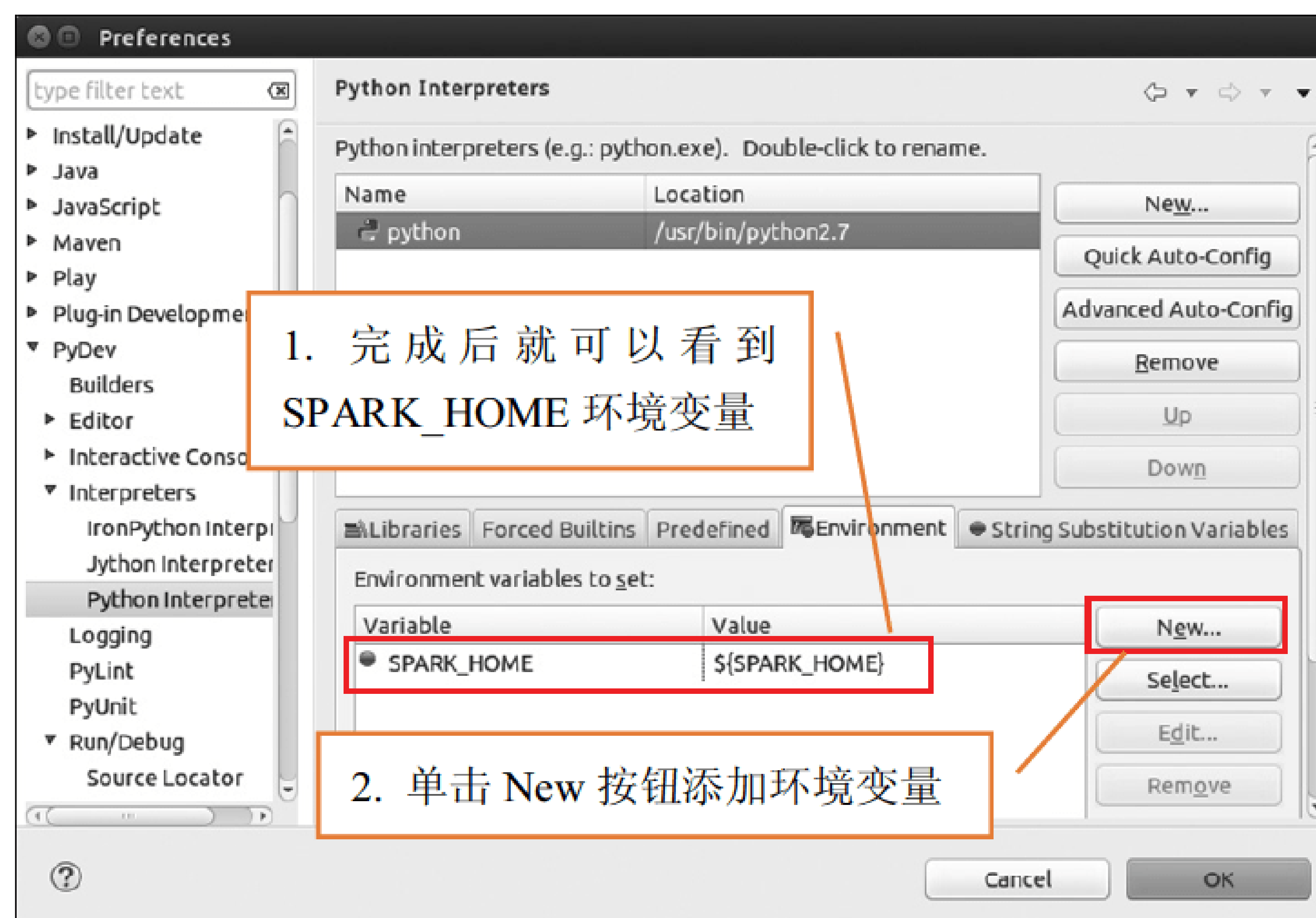


图 11-44　确认已创建 SPARK_HOME 环境变量

步骤 04 添加 HADOOP_CONF_DIR 环境变量（见图 11-45）

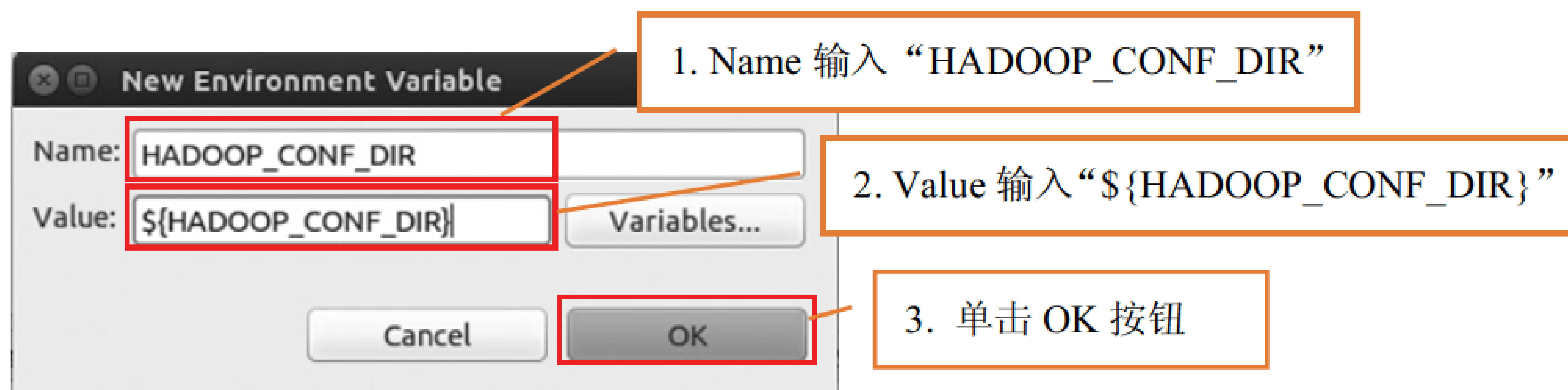


图 11-45　添加 HADOOP_CONF_DIR 环境变量

步骤 05 确认已创建 HADOOP_CONF_DIR 环境变量（见图 11-46）

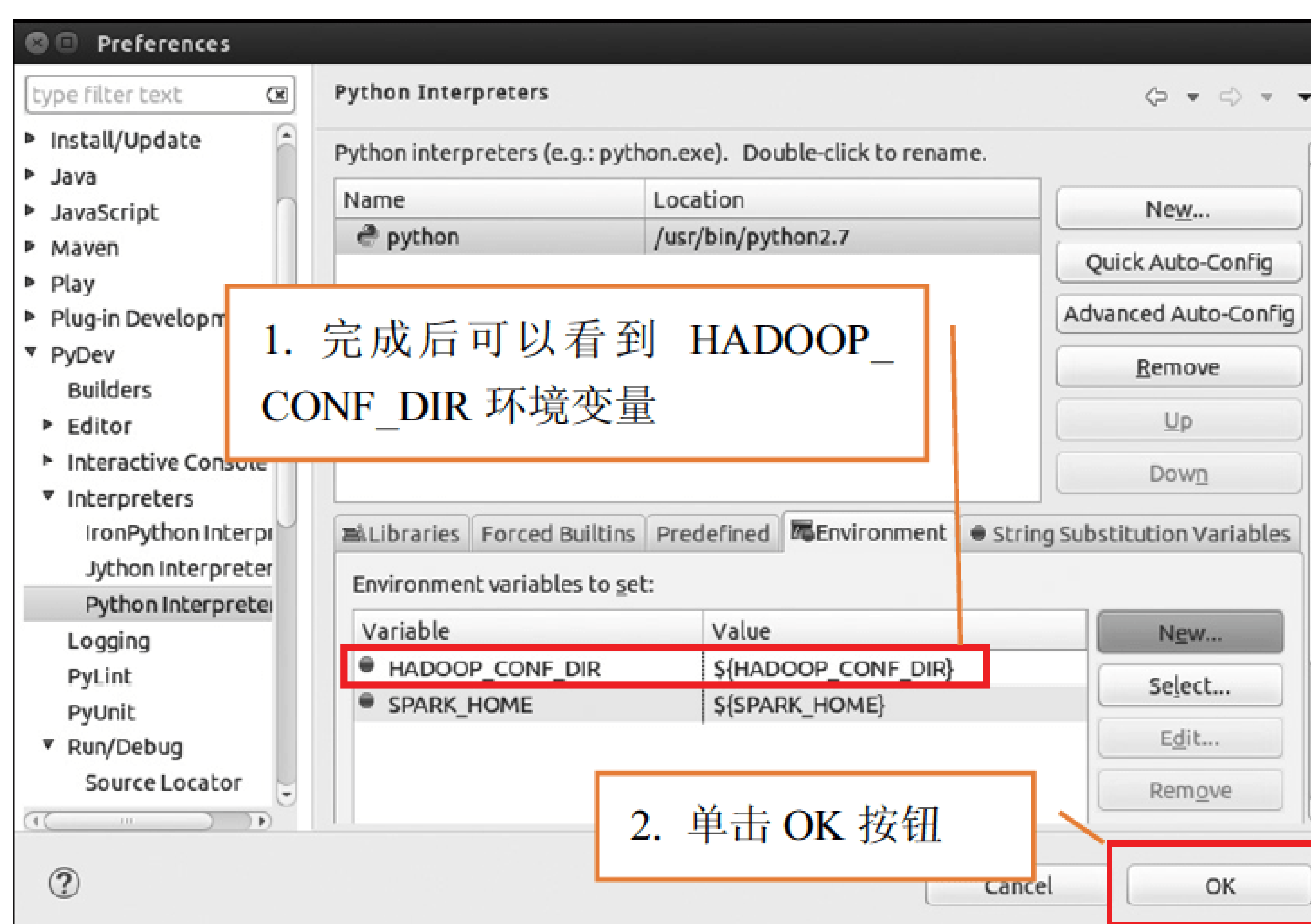


图 11-46　确认已创建 HADOOP_CONF_DIR 环境变量

步骤 06 更新中（见图 11-47）

单击 OK 按钮后就会出现更新界面，如图 11-47 所示。

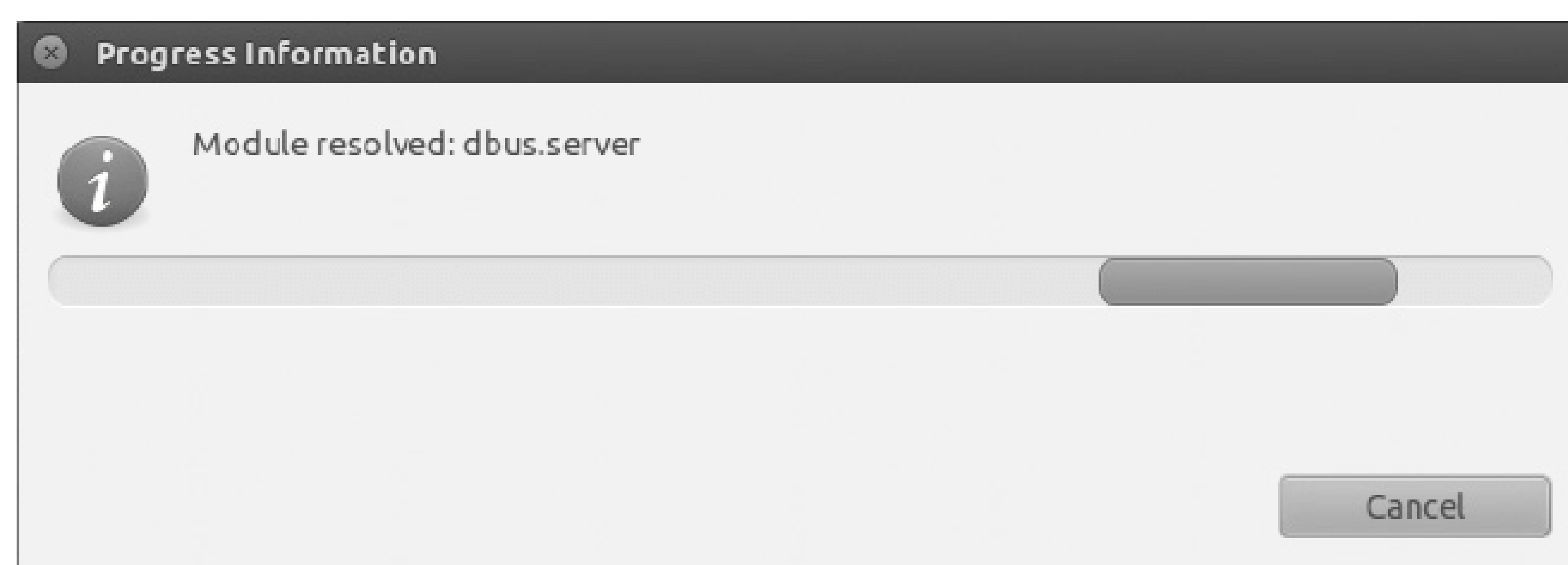


图 11-47　更新内容

11.8 新建 PyDev 项目

之前我们已经完成 PyDev 的设置，接下来我们可以新建 PyDev 项目。

步骤 01 依次选择 File → New → Project... 菜单选项（见图 11-48）

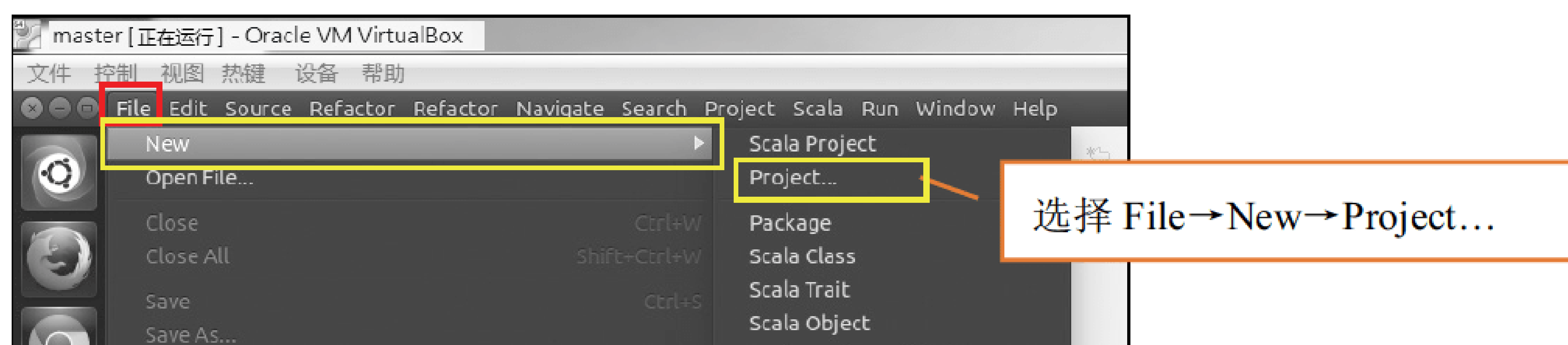


图 11-48　选择新建项目菜单选项

步骤 02 选择 PyDev → PyDev Project（见图 11-49）

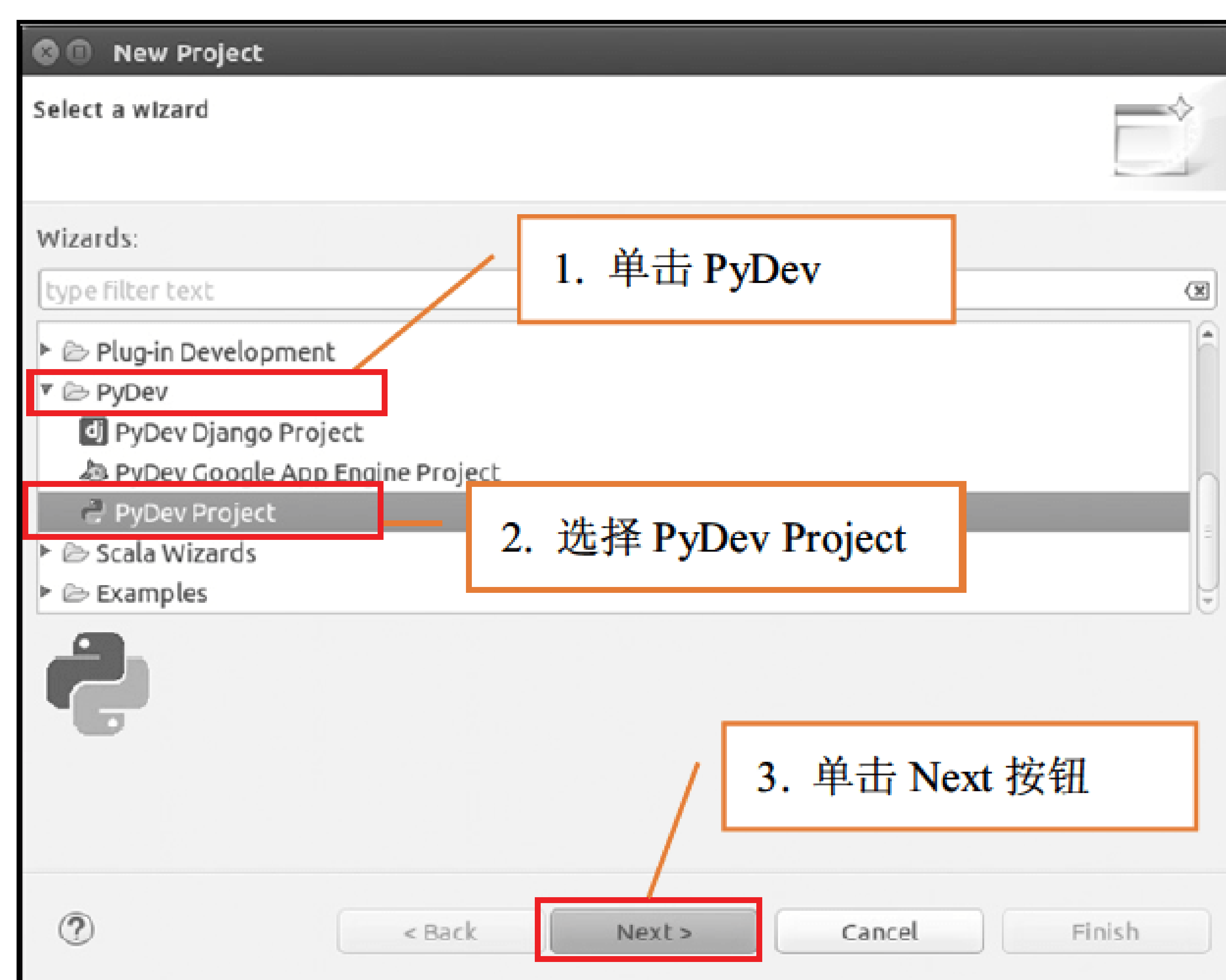


图 11-49　选择新建 PyDev Project 项目

步骤 03 输入项目名称（见图 11-50）

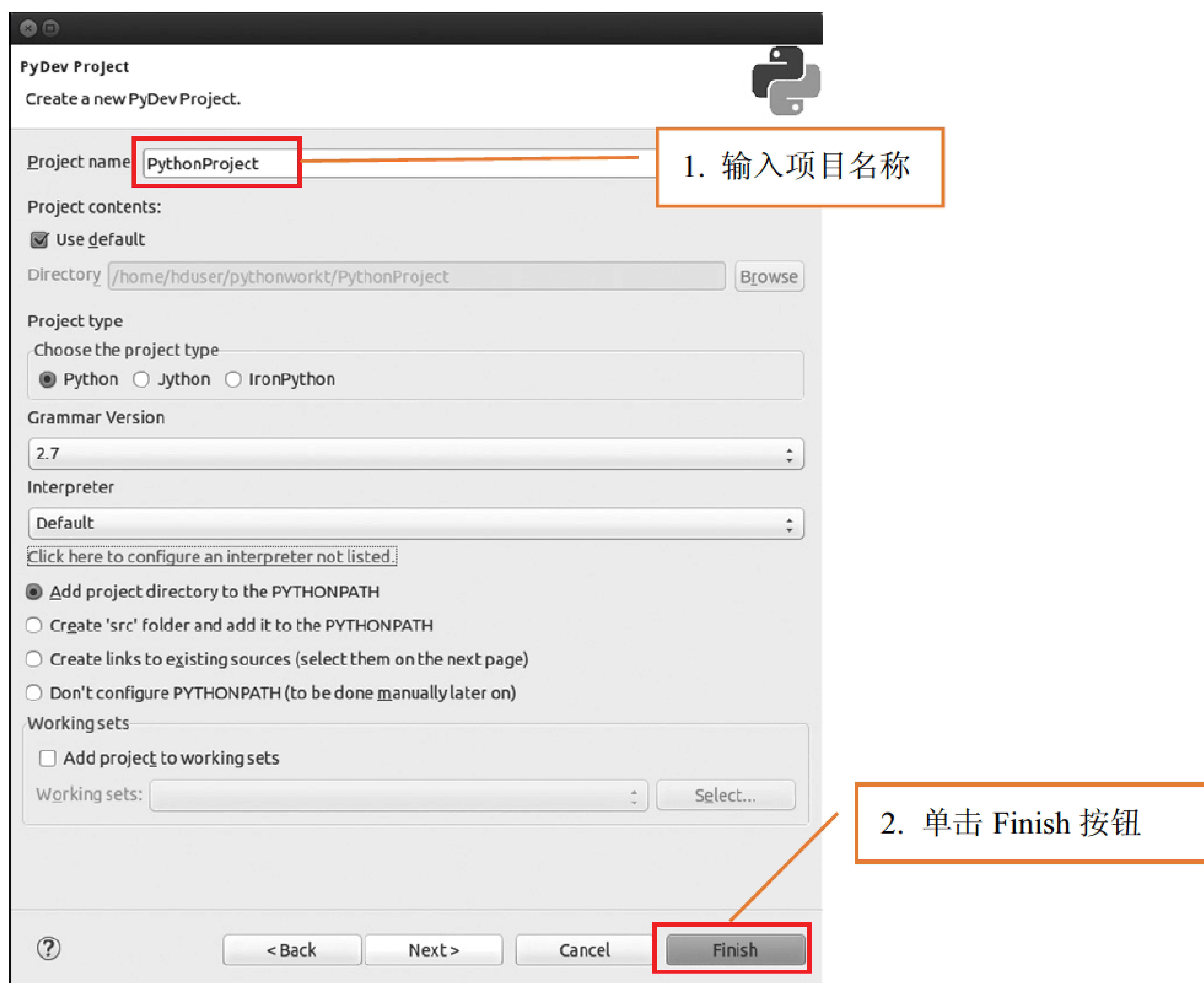


图 11-50 输入项目名称

步骤 04 单击 Yes 按钮（见图 11-51）

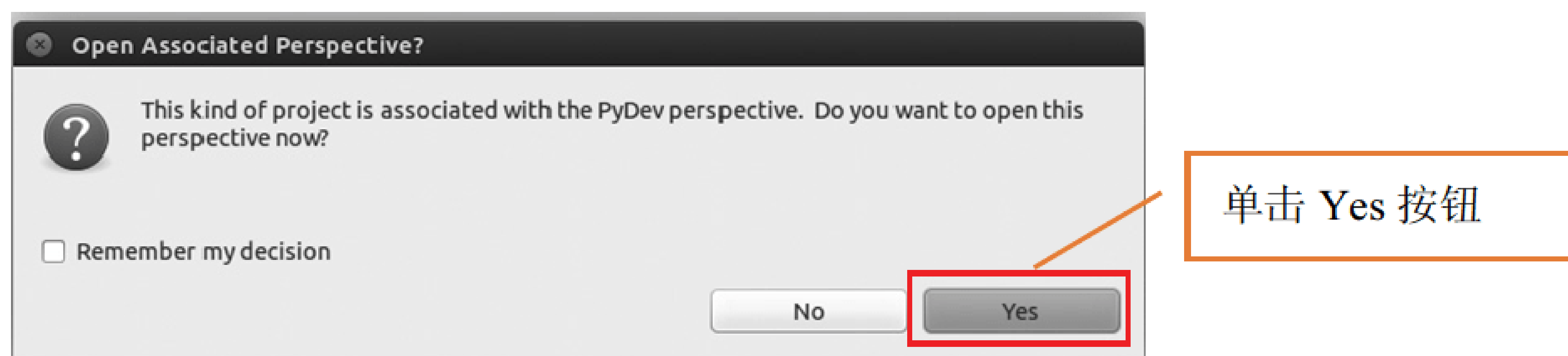


图 11-51

步骤 05 已经创建的项目

完成后就可以看到已经创建的项目，如图 11-52 所示。

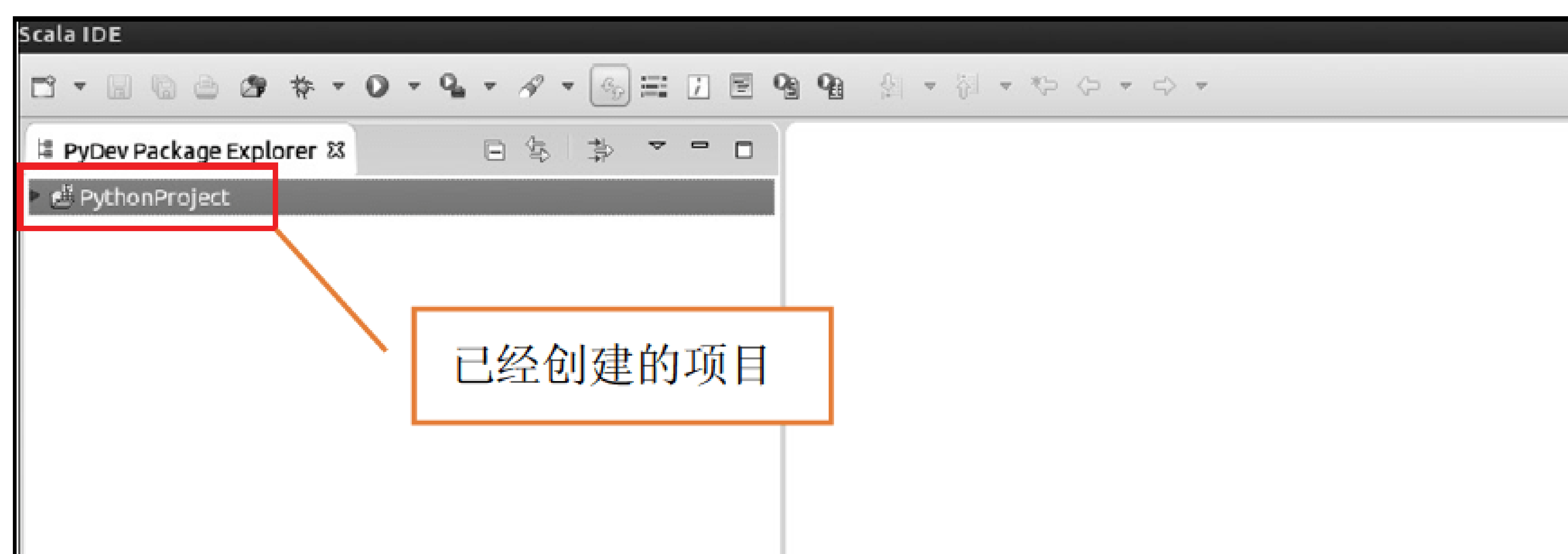


图 11-52 已经创建的项目

11.9 加入 WordCount.py 程序

步骤 01 加入新程序（见图 11-53）

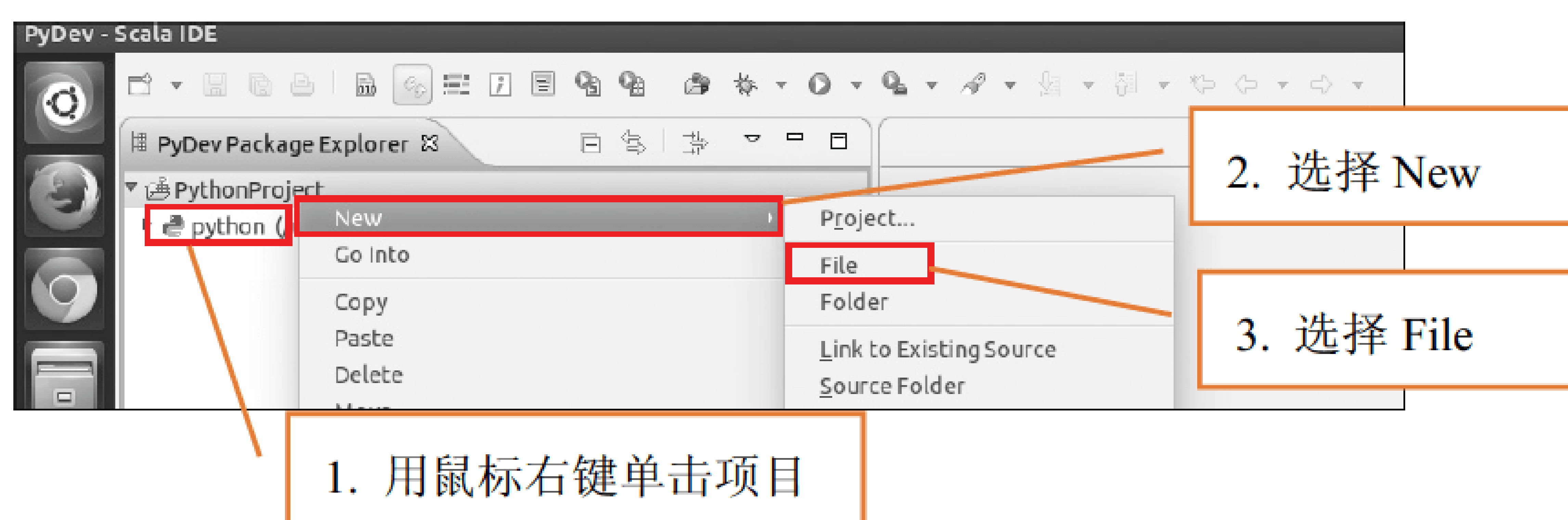


图 11-53 加入新程序

步骤 02 输入程序文件名（见图 11-54）



图 11-54 输入程序文件名

步骤03 已加入的新程序（见图 11-55）

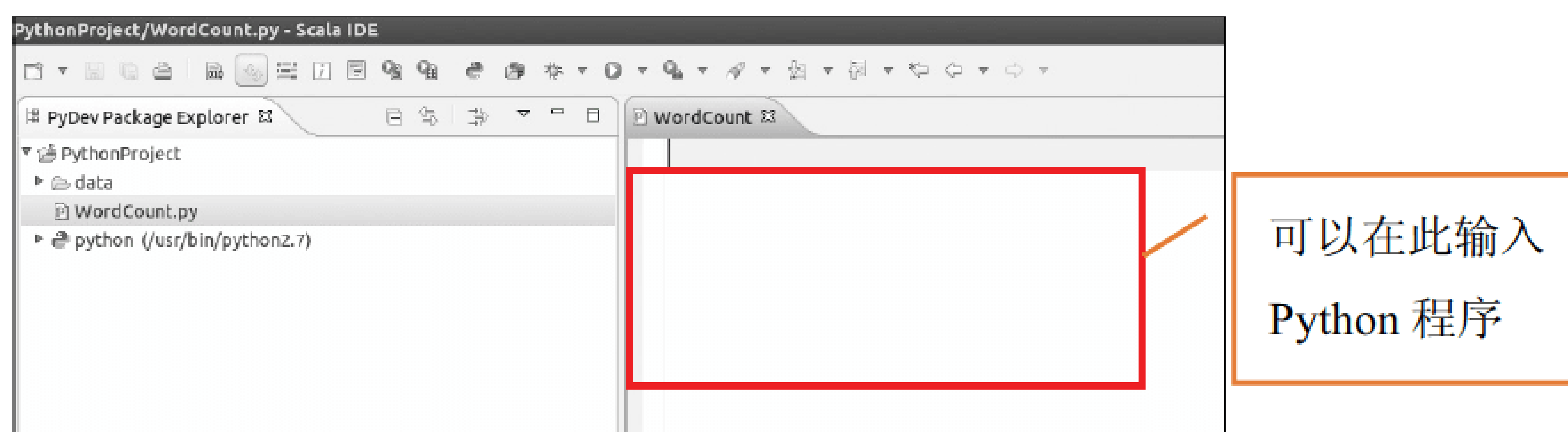


图 11-55 显示已加入的新程序

11.10 输入 WordCount.py 程序

步骤01 导入相关链接库（见表 11-2）

表 11-2 导入相关链接库

语法	说明
# -*- coding: UTF-8 -*-	设置为 utf-8 编码
from pyspark import SparkContext	导入 SparkContext
from pyspark import SparkConf	导入 SparkConf

步骤02 CreateSparkContext()

请参考第 8.1 节介绍。SparkContext 是开发 Spark 应用程序的入口。当我们使用 IPython Notebook 时，可以直接使用 SparkContext。例如，我们可以直接使用 sc.master。可是当我们开发 Python Spark 应用程序时，则必须自行创建 SparkContext。后续所有程序都需要创建 SparkContext，所以我们创建 CreateSparkContext() 函数，以便后重复使用：

```
def CreateSparkContext():
    sparkConf = SparkConf() \
             .setAppName("WordCounts") \
             .set("spark.ui.showConsoleProgress","false")\
    sc = SparkContext(conf = sparkConf)
    print("master="+sc.master)
    SetLogger(sc)
    SetPath(sc)
    return (sc)
```

以上程序代码的 SparkConf 指令很长，为了方便阅读，我们将一条程序指令分为了多行。在 Python 中将程序指令分为多行必须用“\”符号连接指令。以上程序代码的详细说明如表 11-3 所示。

表 11-3　程序代码说明

程序代码	说明
def CreateSparkContext():	定义 CreateSparkContext() 函数
sparkConf = SparkConf()	创建 SparkConf 配置设置对象
.setAppName("WordCounts")	设置 App 名称，此 App 名称会显示在 Spark 或 Hadoop YARN UI 界面
.set("spark.ui.showConsoleProgress","false")	设置不要显示 Spark 执行进度，以免屏幕显示界面太乱
sc = SparkContext(conf = sparkConf)	创建 SparkContext 传入参数： SparkConf 配置设置对象
print ("master="+sc.master)	显示当前运行的模式：local、YARN client 或 Spark Stand alone
SetLogger(sc)	设置不要显示太多信息，在步骤 3 中说明
SetPath(sc)	配置文件读取路径，在步骤 4 中说明
return (sc)	返回 SparkContext sc

步骤 03 设置不要显示太多信息

Spark 程序默认会显示很多信息，这些信息对于调试有帮助。不过信息太多也会让程序执行界面很乱，所以我们编写下列函数，设置不要显示太多信息。

```
def SetLogger( sc ):
    logger = sc._jvm.org.apache.log4j
    logger.LogManager.getLogger("org").setLevel(logger.Level.ERROR)
    logger.LogManager.getLogger("akka").setLevel(logger.Level.ERROR)
    logger.LogManager.getRootLogger().setLevel(logger.Level.ERROR)
```

步骤 04 配置文件读取路径

定义全局变量 Path，配置文件读取路径。

```
def SetPath(sc):
     global Path
     if sc.master[0:5]=="local":
        Path="file:/home/hduser/pythonwork/PythonProject/"
     else:
        Path="hdfs://master:9000/user/hduser/"
```

以上程序判断：

- 如果 sc.master[0:5] 是"local"，代表当前是本地执行，读取本地文件。
- 如果 sc.master[0:5] 不是"local"，就有可能是 YARN Client 或 Spark Stand Alone，必须读取 HDFS 文件。

步骤 05 编写主程序代码

Python 程序中的 __name__可以用来分辨程序是直接执行还是被 import。当我们直接执

行 WordCount 而不是被 import 时，__name__ 变量的值是 "__main__" ，所以会执行下列 if 判断语句之后的主程序代码。

进入主程序代码后，程序会显示开始执行，然后调用 CreateSparkContext() 创建 SparkContext sc。

```
if  name  == " main ":
    print(" 开始执行 RunWordCount")
    sc=CreateSparkContext()
```

步骤 06 读取文本文件

使用 sc.textFile 读取文本文件，在此我们读取本地的 README.md 文本文件作为范例，并显示文本文件的行数。

```
print(" 开始读取文本文件 ...")
textFile = sc.textFile(Path+" data/README.md")
print(" 文本文件共 "+str(textFile.count())+" 行 ")
```

步骤 07 执行 Map/Reduce 运算

执行 Map/Reduce 运算如下：

```
countsRDD = textFile
                      .flatMap(lambda line: line.split(' ')) \
              .map(lambda x: (x, 1))    \
              .reduceByKey(lambda x,y :x+y)
print(" 文字统计共 "+str(countsRDD.count())+" 项数据 ")
```

以上程序代码的详细说明如表 11-4 所示。

表 11-4　程序代码说明

程序代码	说明
countsRDD = textFile	将 textFile 经过一连串 map 与 reduce 运算创建 countsRDD
.flatMap(lambda line: line.split(' '))	用空格符分隔文字，取出每一个英文单词
.map(lambda x: (x, 1))	用 map 转换为 Key-Value (word,1)
.reduceByKey(lambda x,y :x+y)	使用 reduceByKey 将相同的 key 值相加
print(" 文字统计共 "+str(countsRDD.count())+" 项数据 ")	打印 countsRDD 项数

步骤 08 保存文件

countsRDD 使用 saveAsTextFile 将结果保存至本地目录。在此我们加上使用 try catch 语法。如果输出目录已经存在就会发生错误，并显示错误信息。

```
print(" 开始保存至文本文件 ...")
```

```
try:
        countsRDD.saveAsTextFile(Path+ "data/output")
except Exception as e:
        print(" 输出目录已经存在 ，请先删除原有目录 ")
sc.stop()
```

步骤 09　WordCount.py 完成后的屏幕显示界面

程序代码输入后，屏幕显示界面如图 11-56 所示。

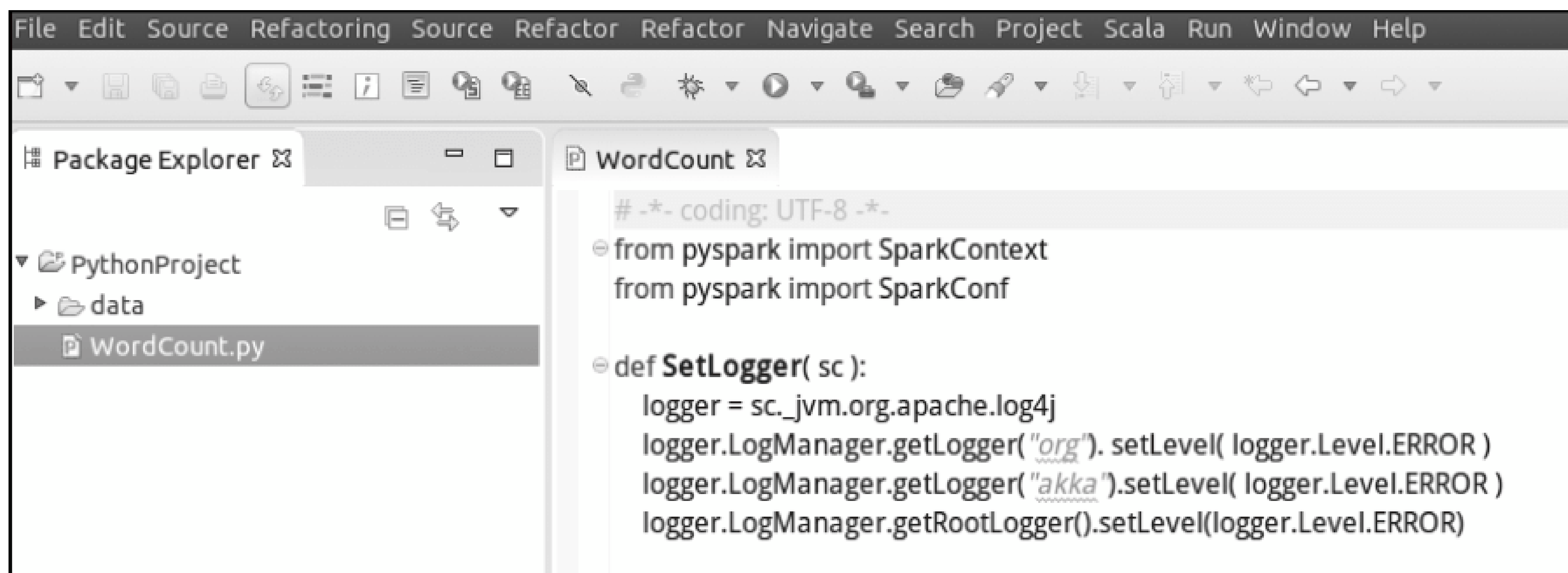


图 11-56　WordCount.py 完成后的界面

11.11 创建测试文件并上传至 HDFS 目录

后续我们将介绍在不同模式运行 WordCount.py，所以我们把文件复制到本地项目目录，再把测试文件上传到 HDFS 目录。

步骤 01　复制本地测试文件

后续我们必须读取文本文件，使用 spark 的 README.md 文件作为测试文件。在“终端”程序中执行下列命令（见图 11-57），首先创建项目目录，然后复制测试文件。

```
hduser@master: ~
hduser@master:~$ mkdir -p ~/pythonwork/PythonProject/data
hduser@master:~$ cp /usr/local/spark/README.md ~/pythonwork/PythonProject/data
hduser@master:~$
```

图 11-57　复制本地测试文件

步骤 02　启动 Hadoop Multi Node Cluster

先启动之前创建的 Hadoop Multi Node Cluster 的所有虚拟机，如图 11-58 所示。



图 11-58　启动 Hadoop Multi Node Cluster

步骤 03 启动 Hadoop cluster

在 master 服务器的“终端”程序中输入下列命令：

➢ **启动 Hadoop 群集 cluster**

```
start-all.sh
```

执行后屏幕显示界面如图 11-59 所示。

```
hduser@master: ~
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-master.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data2.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-mast
er.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-master.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data1.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data3.out
```

图 11-59　启动 Hadoop cluster

步骤 04 复制测试文件到 HDFS

我们必须先将输入文件复制到 HDFS，可在“终端”程序中输入下列命令：

➢ **创建 HDFS 测试目录**

```
hadoop fs -mkdir -p /user/hduser/data
```

➢ **查看 HDFS 测试文件**

```
hadoop fs -ls /user/hduser/data/README.md
```

执行后屏幕显示界面如图 11-60 所示。

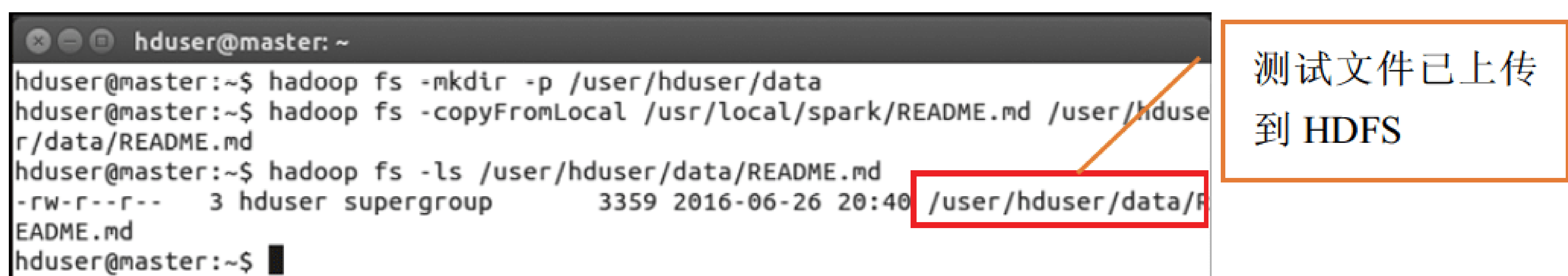


图 11-60　复制测试文件到 HDFS

11.12 使用 spark-submit 执行 WordCount 程序

之前我们已经编辑完成 WordCount.py 程序。接下来我们将在“终端”程序界面用 spark-submit 执行 Python Spark application。

步骤 01 spark-submit 详细介绍

spark-submit 常用的选项如 11-5 所示。

表 11-5　spark-submit 常用选项

选项	说明
--master MASTER_URL	可设置 Spark 在什么环境运行
--driver-memory MEM	driver 程序所使用的内存
--executor-memory MEM	executor 程序所使用的内存
--name NAME	要运行的 application 名称，此名称后续会显示在 Hadoop 或 Spark Web UI 界面中
Python 程序文件名	要运行的 Python 程序

--master MASTER_URL 选项可设置 Spark 在什么环境中运行，如表 11-6 所示。

表 11-6　--master MASTER_URL 选项说明

MASTER URL	说明
Local	在本地运行，只使用一个线程
local[K]	在本地运行，使用 K 个线程（会使用本地计算机的多核 CPU）
local[*]	在本地运行，Spark 会自动尽量利用本地计算机上的多核 CPU
spark://HOST:PORT	在 Spark Standalone Cluster 上运行，例如 spark://master:7077（默认 port 是 7077）
mesos://HOST:PORT	在 Mesos cluster 上运行（默认 port 是 5050）

（续表）

MASTER URL	说明
YARN	在 YARN Client 上运行，必须要设置 HADOOP_CONF_DIR 或 YARN_CONF_DIR 环境变量

步骤02 在 local 运行 WordCount

在“终端”程序中输入下列命令，在 local 运行 WordCount：

➢ **切换到 WordCount 项目目录**

```
cd ~/pythonwork/PythonProject
```

➢ **在 local 执行 WordCount**

```
spark-submit --driver-memory 2g --master local[4] WordCount.py
```

运行后屏幕显示界面如图 11-61 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~$ cd ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ spark-submit --driver-memory 2g --mast
er local[4] WordCount.py
开始运行RunWordCount
16/08/23 13:23:56 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
master=local[4]
开始读取文本文件...
文本文件共95行
文字统计共260项数据
开始保存到文本文件...
```

图 11-61 在 local 运行 WordCount

spark-submit 命令的详细说明如表 11-7 所示。

表 11-7 spark-submit 命令说明

命令	说明
spark-submit	spark-submit 命令
--driver-memory 2g	设置 driver 程序使用 2GB 的内存
--master local[4]	在本地运行，使用 4 个线程（会使用本地计算机上的多核 CPU）
WordCount.py	WordCount Python 程序

步骤03 查看输出文件目录

运行完成后，就可以看到输出的文件内容。在“终端”程序中输入下列命令：

➢ **查看输出文件目录**

```
ll data/output
```

运行后屏幕显示界面如图 11-62 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ ll data/output
总用量 20
drwxrwxr-x 2 hduser hduser 4096  7月  9 20:16 ./
drwxrwxr-x 4 hduser hduser 4096  8月 19 13:37 ../
-rwxrwxr-x 1 hduser hduser 4496  7月  9 20:16 part-00000*
-rwxrwxr-x 1 hduser hduser   44  7月  9 20:16 .part-00000.crc*
-rwxrwxr-x 1 hduser hduser    0  7月  9 20:16 _SUCCESS*
```

图 11-62　查看输出文件目录

步骤 04 查看输出文件内容

在“终端”程序中输入下列命令：

➢ **查看输出文件内容**

```
cat data/output/part-00000|more
```

运行后屏幕显示界面如图 11-63 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ cat data/output/part-00000|more
(u'', 67)
(u'help', 1)
(u'when', 1)
(u'Hadoop', 3)
(u'"local"', 1)
(u'including', 3)
(u'computation', 1)
(u'file', 1)
(u'high-level', 1)
(u'find', 1)
(u'web', 1)
(u'Shell', 2)
(u'cluster', 2)
(u'also', 4)
```

图 11-63　查看输出文件内容

11.13 在 Hadoop YARN-client 上运行 WordCount 程序

接下来要介绍如何使用 spark-summit 在 Hadoop Yarn 上运行 WordCount Spark 程序。

步骤 01 在 Hadoop Yarn 上运行 WordCount 程序

在“终端”程序中输入下列命令，在 Hadoop Yarn 上运行 WordCount：

➢ **切换至 WordCount 项目目录**

```
cd ~/pythonwork/PythonProject
```

➢ **在 Hadoop Yarn 上运行 WordCount**

```
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop spark-submit --driver-memory 512m
```

```
--executor-cores 2 --master yarn --deploy-mode client WordCount.py
```

运行后屏幕显示界面如图 11-64 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~$ cd ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ HADOOP_CONF_DIR=/usr/local/hadoop/etc/
hadoop spark-submit --driver-memory 512m --executor-cores 2 --master yarn --depl
oy-mode client WordCount.py
开始运行RunWordCount
16/08/24 18:33:17 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
16/08/24 18:33:21 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is
 set, falling back to uploading libraries under SPARK_HOME.
master=yarn
开始读取文本文件...
文本文件共95行
文字统计共260项数据
开始保存到文本文件...
```

目前是 YARN-client 模式

图 11-64　在 YARN-client 模式运行程序

以上 spark-submit 命令的详细说明如表 11-8 所示。

表 11-8　spark-submit 命令说明

命令	说明
HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop	在 Hadoop YARN 运行 Spark 应用程序时必须先设置 HADOOP_CONF_DIR 环境变量，请设置为 Hadoop 配置文件目录
spark-submit	spark-submit 命令
--driver-memory 512m	设置 driver 程序使用 512MB 的内存
--executor-cores 2	设置可执行的 CPU
--master YARN	在 Hadoop YARN 上运行
--deploy-mode client	部署的方式为 client
WordCount.py	要运行的 Python 程序

步骤 02　查看执行完成后 HDFS 产生的目录

运行后，输出文件产生在 HDFS data/output 目录中。在“终端”程序中输入下列命令：

```
hadoop fs -ls /user/hduser/data/output
```

运行后屏幕显示界面如图 11-65 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ hadoop fs -ls /user/hduser/data/output
Found 3 items
-rw-r--r--   3 hduser supergroup          0 2016-08-23 14:20 /user/hduser/data/output/_SUCCESS
-rw-r--r--   3 hduser supergroup       2352 2016-08-23 14:20 /user/hduser/data/output/part-00000
-rw-r--r--   3 hduser supergroup       2144 2016-08-23 14:20 /user/hduser/data/output/part-00001
```

图 11-65　查看执行完成后 HDFS 产生的目录

步骤03 查看运行完成后 HDFS 产生的文件

在“终端”程序中输入下列命令，查看输出文件的内容。查看 HDFS 的 data/output/part-00000 文件：

```
hadoop fs -cat /user/hduser/data/output/part-00000|more
```

运行后屏幕显示界面如图 11-66 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ hadoop fs -cat /user/hduser/data/output/part-00000|more
(u'', 67)
(u'storage', 1)
(u'"local"', 1)
(u'including', 3)
(u'computation', 1)
(u'file', 1)
(u'using:', 1)
(u'guidance', 2)
(u'Scala,', 1)
(u'environment', 1)
(u'only', 1)
(u'rich', 1)
(u'Apache', 1)
```

图 11-66 查看运行完成后 HDFS 产生的文件

步骤04 在 Hadoop Web 界面查看 WordCounts

已经在 Hadoop YARN 执行 WordCount 之后，我们就可以在 Hadoop Web 界面看到这个应用程序（Application）。可按照如图 11-67 所示的步骤打开 Hadoop Web 界面。

图 11-67 在 Hadoop Web 界面查看 WordCount

WordCounts 这个应用程序名称就是我们在第 11.10 节.setAppName("WordCounts")中设置的。

步骤05 有时运行时会产生错误信息

有时运行时会发生错误信息，再运行一次即可，如图 11-68 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~$ cd ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ spark-submit --driver-memory 1024m --e
xecutor-cores 2 --executor-memory 1024m --master yarn --deploy-mode client WordC
ount.py
开始运行RunWordCount
16/08/23 13:33:56 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
16/08/23 13:34:03 ERROR SparkContext: Error initializing SparkContext.
org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.hdfs.server.namenode.Saf
eModeException): Cannot create directory /user/hduser/.sparkStaging/application_
1471930404407_0001. Name node is in safe mode.
The reported blocks 0 needs additional 34 blocks to reach the threshold 0.9990 o
f total blocks 34.
The number of live datanodes 0 has reached the minimum number 0. Safe mode will
be turned off automatically once the thresholds have been reached.
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkNameNodeSafe
Mode(FSNamesystem.java:1366)
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInt(FSNames
ystem.java:4258)
```

图 11-68 运行发生错误时再运行一次

11.14 在 Spark Standalone Cluster 上运行 WordCount 程序

接下来要介绍如何使用 spark-summit 在 Standalone Cluster 上运行 WordCount Spark 程序。如果尚未构建 Spark Standalone Cluster，可以参考第 8.9 节的内容构建 Spark Standalone Cluster 运行环境。

步骤 01 删除已产生的目录

我们后续还要继续测试，所以在“终端”程序中输入下列命令：

➢ **删除已产生目录**

删除 HDFS 的 data/output 目录，加上-R 会删除所有子目录与文件。

```
hadoop fs -rm -R /user/hduser/data/output
```

运行后屏幕显示界面如图 11-69 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ hadoop fs -rm -R /user/hduser/data/output
16/08/23 14:23:08 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes,
Emptier interval = 0 minutes.
Deleted /user/hduser/data/output
```

图 11-69 删除已产生目录

步骤 02 启动 Standalone Cluster

在“终端”程序中输入下列命令：

➢ **启动 Standalone Cluster**

```
/usr/local/spark/sbin/start-all.sh
```

运行后屏幕显示界面如图 11-70 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ /usr/local/spark/sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs
/spark-hduser-org.apache.spark.deploy.master.Master-1-master.out
data2: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data2.out
data3: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data3.out
data1: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spa
rk/logs/spark-hduser-org.apache.spark.deploy.worker.Worker-1-data1.out
```

图 11-70　启动 Standalone Cluster

步骤 03 在 Spark Standalone Cluster 上运行 WordCount 程序

在“终端”程序中输入下列命令，就可以在 Spark Standalone Cluster 运行 WordCount 程序：

➢ **切换至 WordCount 项目目录**

```
cd ~/pythonwork/PythonProject/
```

➢ **在 Spark Standalone Cluster 上运行 WordCount 程序**

```
spark-submit --master spark://master:7077 --deploy-mode client --executor-memory
500M --deploy-mode client --total-executor-cores 2 WordCount.py
```

运行后屏幕显示界面如图 11-71 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~$ cd ~/pythonwork/PythonProject/
hduser@master:~/pythonwork/PythonProject$ spark-submit --master spark://master:7077
--deploy-mode client --executor-memory 500M --total-executor-cores 2 wordcount.py
开始运行RunWordCount
16/03/10 22:26:04 WARN NativeCodeLoader: Unable to load native-hadoop library for yo
ur platform... using builtin-java classes where applicable
master=spark://master:7077
开始读取文本文件...
文本文件共95行
文字统计共260项数据
开始保存到文本文件...
hduser@master:~/pythonwork/PythonProject$
```

目前是 Spark Standalone Cluster 模式

图 11-71　运行 WordCount 程序

spark-submit 命令的详细说明如表 11-9 所示。

表 11-9　命令说明

命令	说明
spark-submit	spark-submit 命令
--master spark://master:7077	在 Spark Standalone Cluster 运行
--deploy-mode client	部署模式为 client

（续表）

命令	说明
--executor-memory 500M	设置 driver 程序使用 500MB 的内存
-total-executor-cores 2	设置运行的 CPU 数
--class RunWordCount	设置 main 类
WordCount.py	要运行的 python 程序

步骤 04 查看程序运行后的输出目录

在“终端”程序中输入下列命令：

➢ **查看产生在 HDFS data/output 目录中的输出文件**

```
hadoop fs -ls /user/hduser/data/output
```

运行后屏幕显示界面如图 11-72 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ hadoop fs -ls /user/hduser/data/output
Found 3 items
-rw-r--r--   3 hduser supergroup          0 2016-08-23 14:24 /user/hduser/data/output/_SUCCESS
-rw-r--r--   3 hduser supergroup       2352 2016-08-23 14:24 /user/hduser/data/output/part-00000
-rw-r--r--   3 hduser supergroup       2144 2016-08-23 14:24 /user/hduser/data/output/part-00001
hduser@master:~/pythonwork/PythonProject$
```

图 11-72　查看产生在 HDFS data/output 目录中的输出文件

步骤 05 Spark Standalone Web UI 界面

运行后，我们可以在 Spark Standalone Web UI 界面查看程序的运行记录与状态，在浏览器输入下列网址：

➢ **查看 Spark Standalone Web UI 界面**

```
http://master:8080/
```

运行后屏幕显示界面如图 11-73 所示。我们可以看到启动后共有 3 个 Worker 以及已运行完毕的程序 WordCounts。



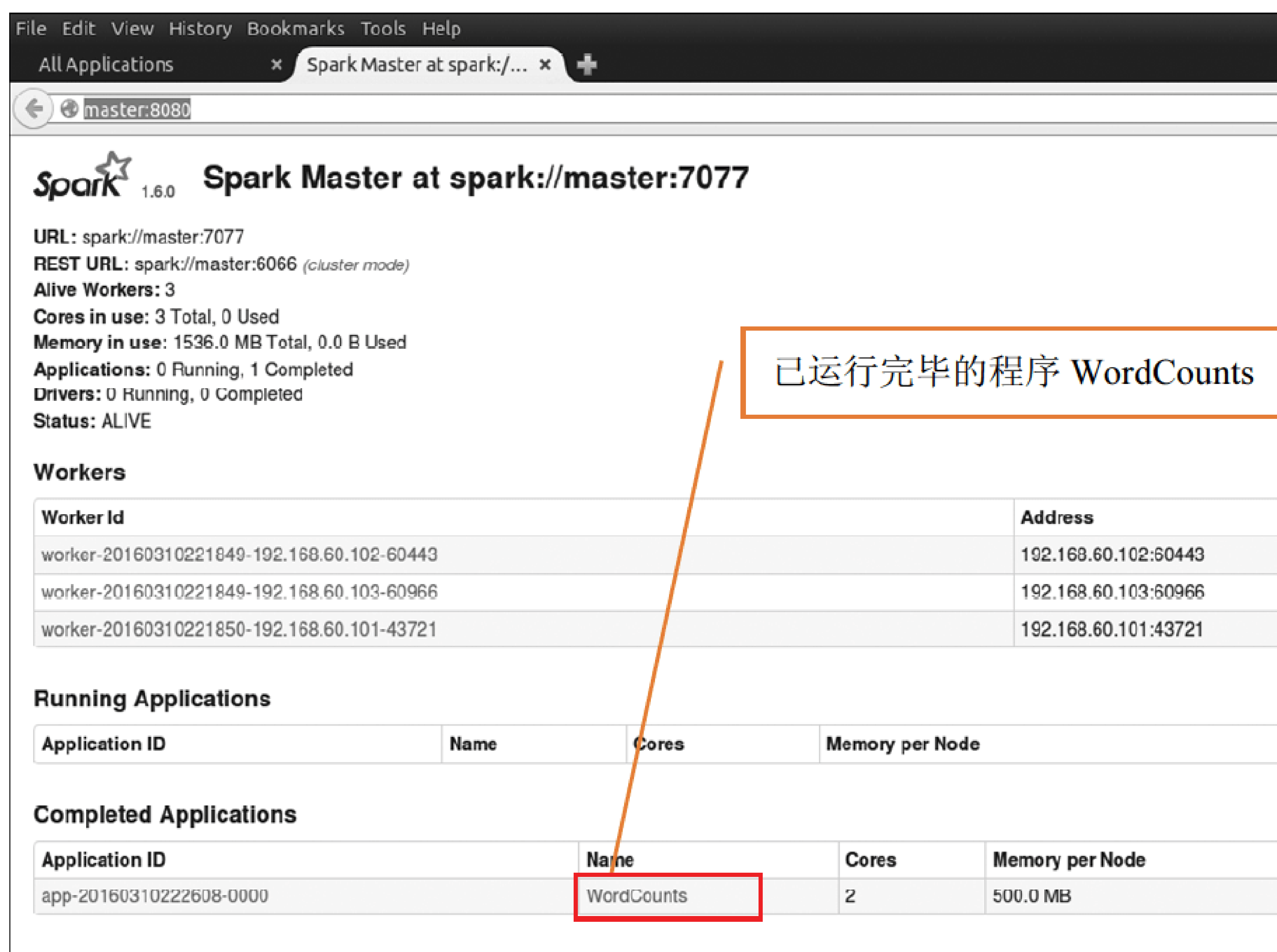


图 11-73　查看 Spark Standalone Web UI 界面

步骤 06　删除已产生的目录

我们后续还要继续测试，所以在“终端”程序中输入下列命令：

➢　**删除输出目录**

删除 HDFS 的 data/output 目录，加上-R 会删除所有子目录与文件。

```
hadoop fs -rm -R /user/hduser/data/output
```

运行后屏幕显示界面如图 11-74 所示。

```
hduser@master: ~/pythonwork/PythonProject
hduser@master:~/pythonwork/PythonProject$ hadoop fs -rm -R /user/hduser/data/output
16/08/23 14:23:08 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes,
Emptier interval = 0 minutes.
Deleted /user/hduser/data/output
```

图 11-74　删除输出目录

11.15 在 eclipse 外部工具运行 Python Spark 程序

在之前的章节中，我们介绍可以在“终端”程序中使用spark-submit 运行 Python Spark 程

序，并且可以运行在local、Hadoop YARN 与Spark Standalone 模式。可是如果每次运行都要在“终端”程序中运行，较不方便。

在 eclipse 我们可以使用外部工具以 spark-submit 来运行 Python Spark 程序，并且也可以运行在 local、Hadoop YARN 与 Spark Standalone 模式。这种方式提高了 eclipse 开发 Python Spark 的效率，只需要在编辑界面修改好程序、运行外部工具即可。修改与运行都在同一个界面。

步骤01 外部工具的设置

可以使用下列两种方式（见图 11-75），进入外部工具的设置：

- **菜单：**Run → External Tools → External Tools Configurations...

或

- **单击**图标 → External Tools Configurations.

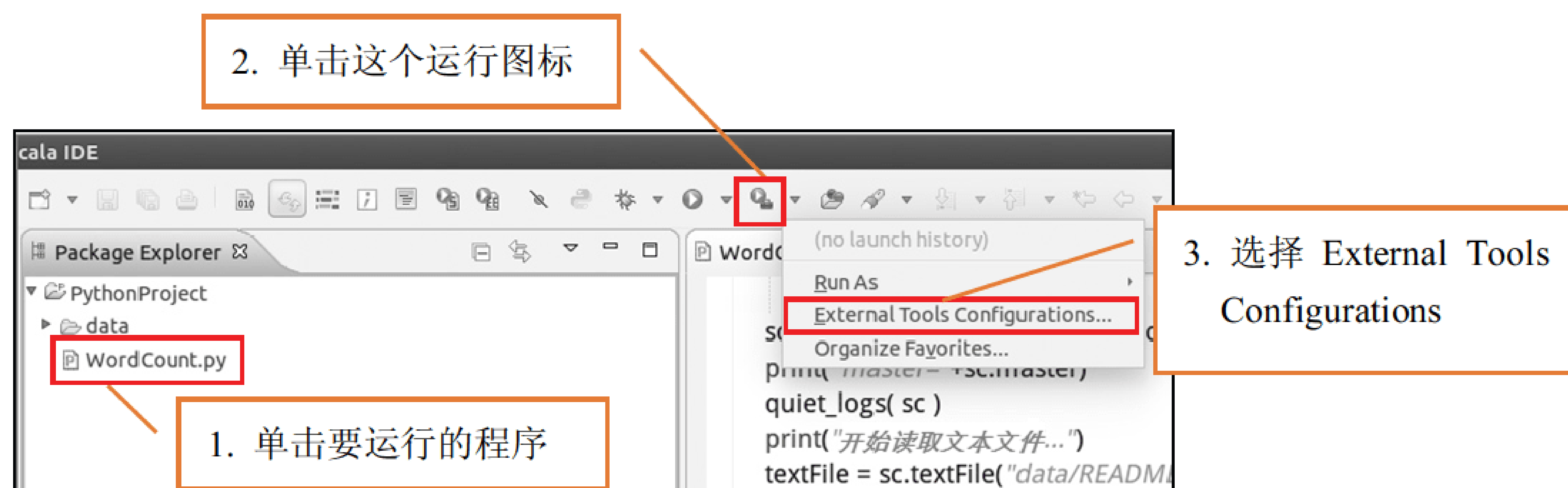


图 11-75 选择设置外部工具命令

步骤02 新建外部工具的设置（见图 11-76）



图 11-76 新建项目

步骤03 设置外部工具

经过上一步骤新建外部工具的设置，新建了一个 New_configuration，然后设置外部工具

参数，如图 11-77 所示。

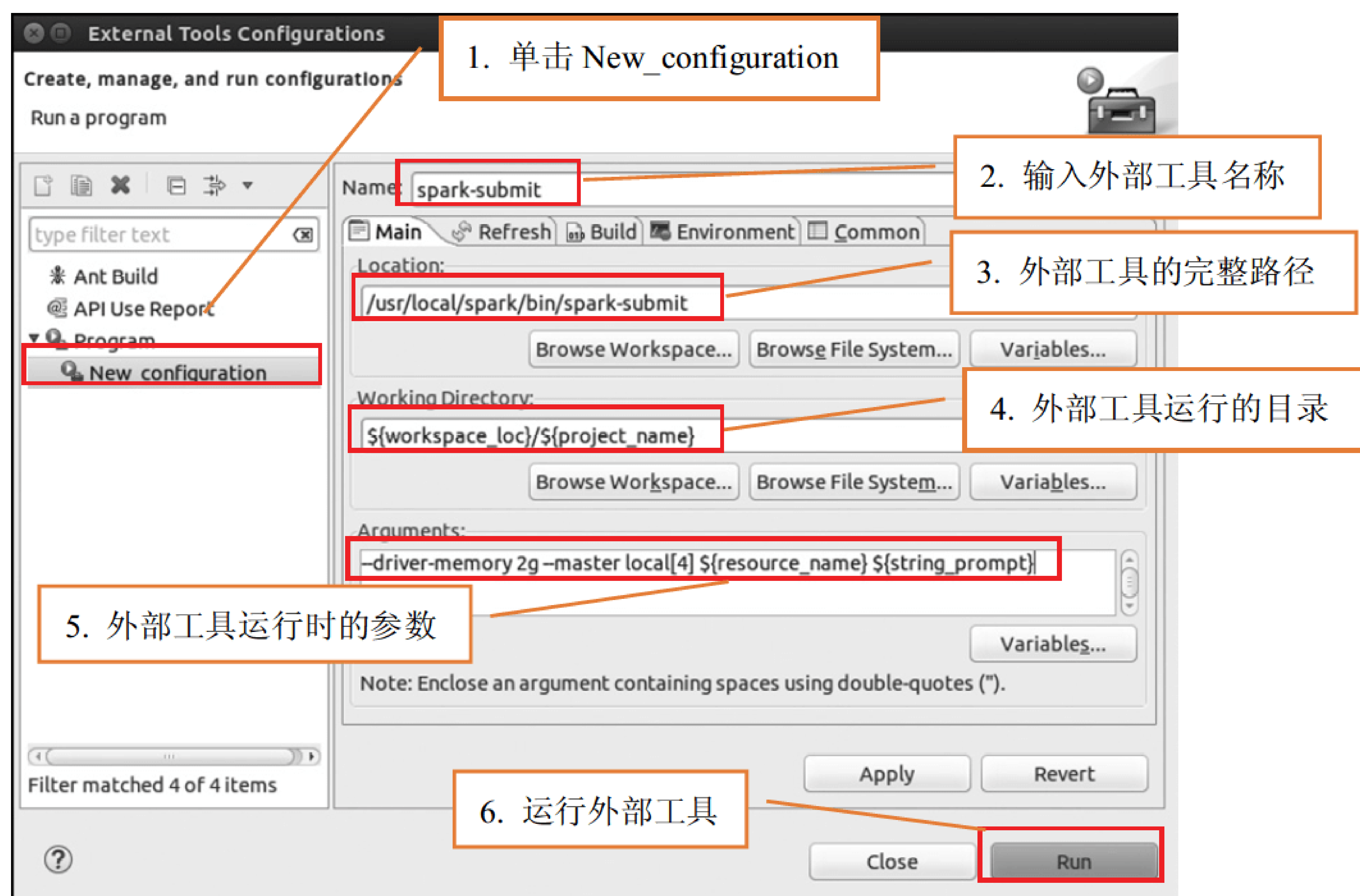


图 11-77　设置外部工具参数

以上设置说明如表 11-10 所示。

表 11-10　外部工具参数设置说明

设置项目	设置值与说明
name	spark-submit
	外部工具的名称，会显示在运行外部工具的菜单上
location	/usr/local/spark/bin/spark-submit
	外部工具的路径，在本范例是 spark-submit 程序的完整路径
Working Directory	${workspace_loc}/${project_name}
	设置外部工具运行的目录 • ${workspace_loc} 代表 eclipse 的工作目录，在本范例中是 /home/hduser/pythonwork • ${project_name} 代表项目名称，在本范例中是 PythonProject 合起来后 /home/hduser/pythonwork/PythonProject 就是工作目录
Arguments	--driver-memory 2g --master local[4] ${resource_name} ${string_ prompt}
	运行外部工具的参数，其中： • ${resource_name} 是当前选中的文件，也就是 WordCount.py • ${string_prompt} 是程序运行时输入的参数

步骤04 spark-submit 设置环境变量

因为后续我们的范例程序会使用到一些 Python 链接库，例如 numpy、pandas 等，所以必须添加设置环境变量 PYSPARK_PYTHON。在这里我们一样使用字符串替代变量（见图 11-78），简化设置过程。

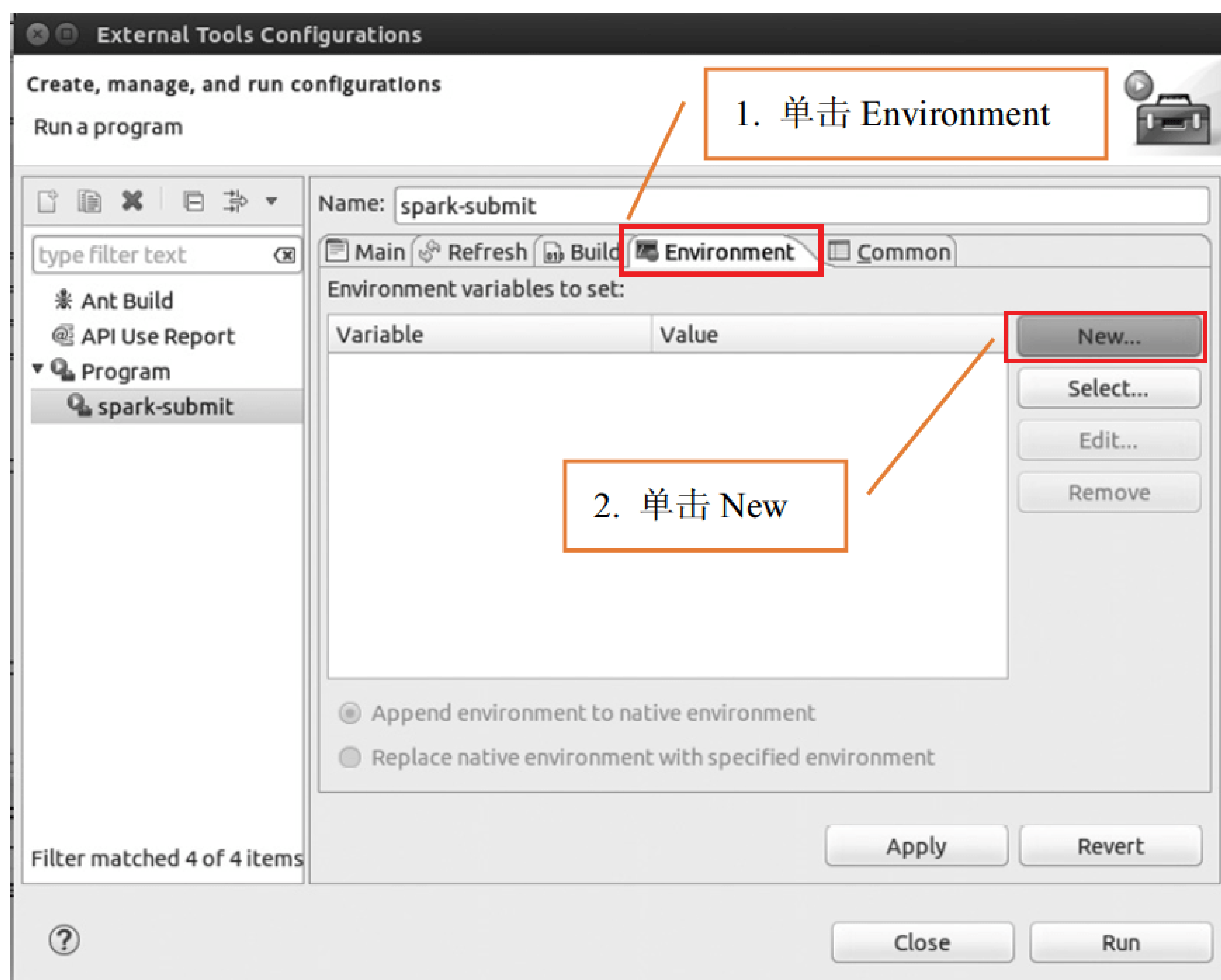


图 11-78 spark-submit 设置环境变量

步骤05 新建 PYSPARK_PYTHON 环境变量（见图 11-79）

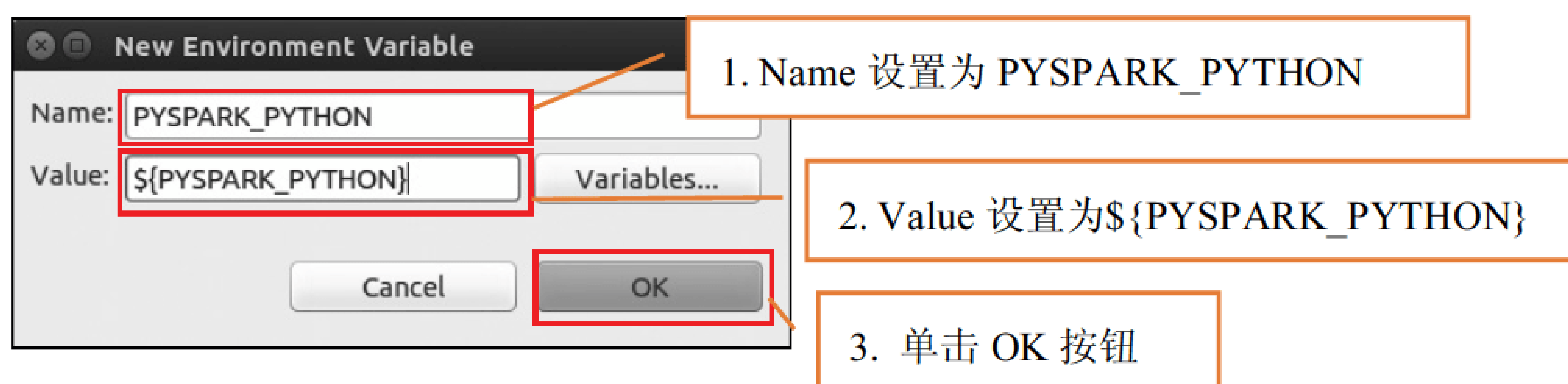


图 11-79 新建 PYSPARK_PYTHON 环境变量

步骤06 运行外部工具（见图 11-80）



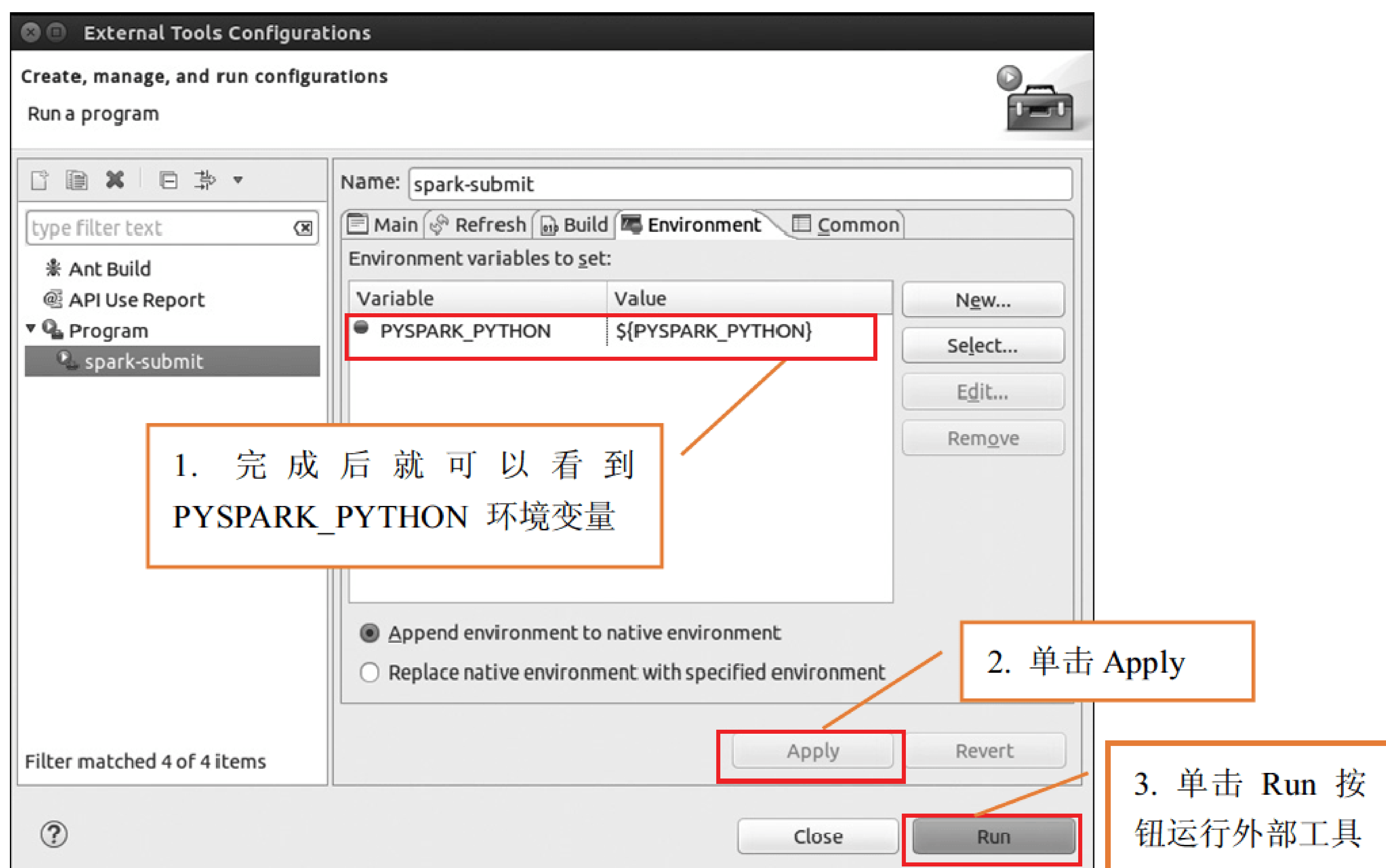


图 11-80　运行外部工具

步骤 07　不要输入参数，直接单击 OK 按钮

因为之前在步骤 3 中设置外部工具时，Arguments 已设置为 ${string_prompt}，所以运行后系统会要求输入程序运行时的参数。由于 WordCount.py 不需要输入参数，因此在这里不要输入参数，直接单击 OK 按钮，如图 11-81 所示。

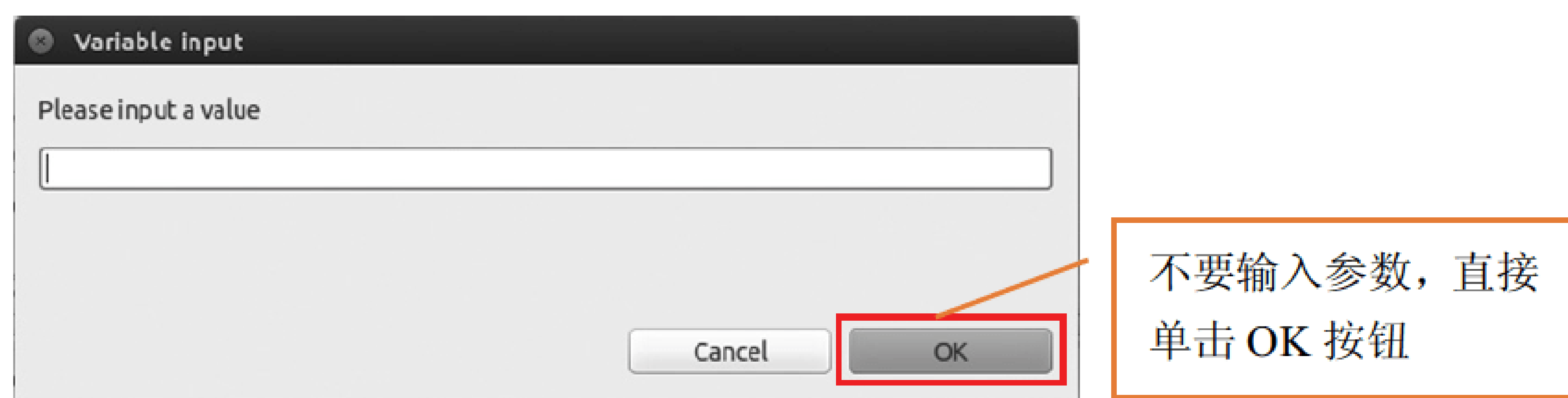


图 11-81　不输入参数

步骤 08　外部工具运行的结果（见图 11-82）

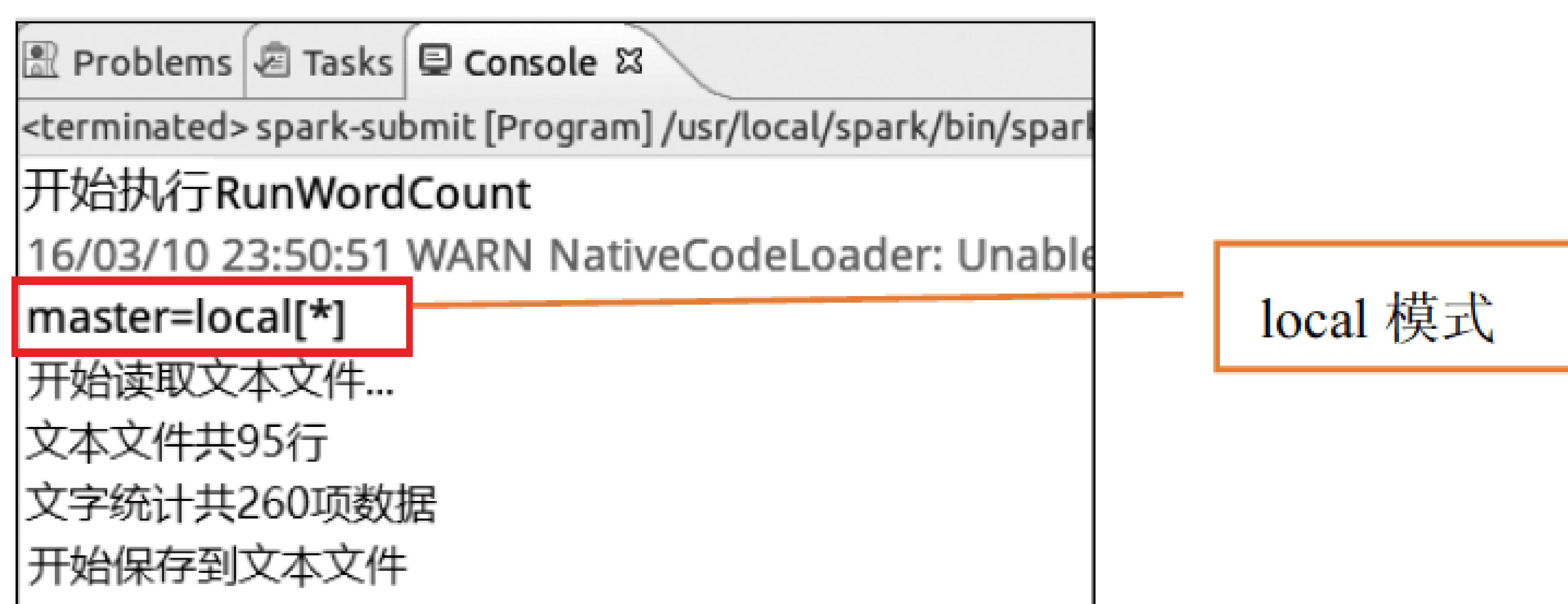


图 11-82　外部工具运行的结果

步骤09 查看输出的 data/output 目录

当运行完 WordCount 之后，就会在 data/output 目录中产生输出文件，但是在 eclipse 中还看不到内容。我们可以在 Package Explorer 中先单击 data 目录，然后按 F5 键更新目录中的内容，随后就可以看到产生的目录。展开 output 目录之后会看到输出的文件。

如果输出的文件很大，saveAsTextFile 命令会自动分为多个文件。本范例就会产生 3 个文件：_SUCCESS 代表输出成功，part-00000 与 part-00001 两个数据文件，如图 11-83 所示。

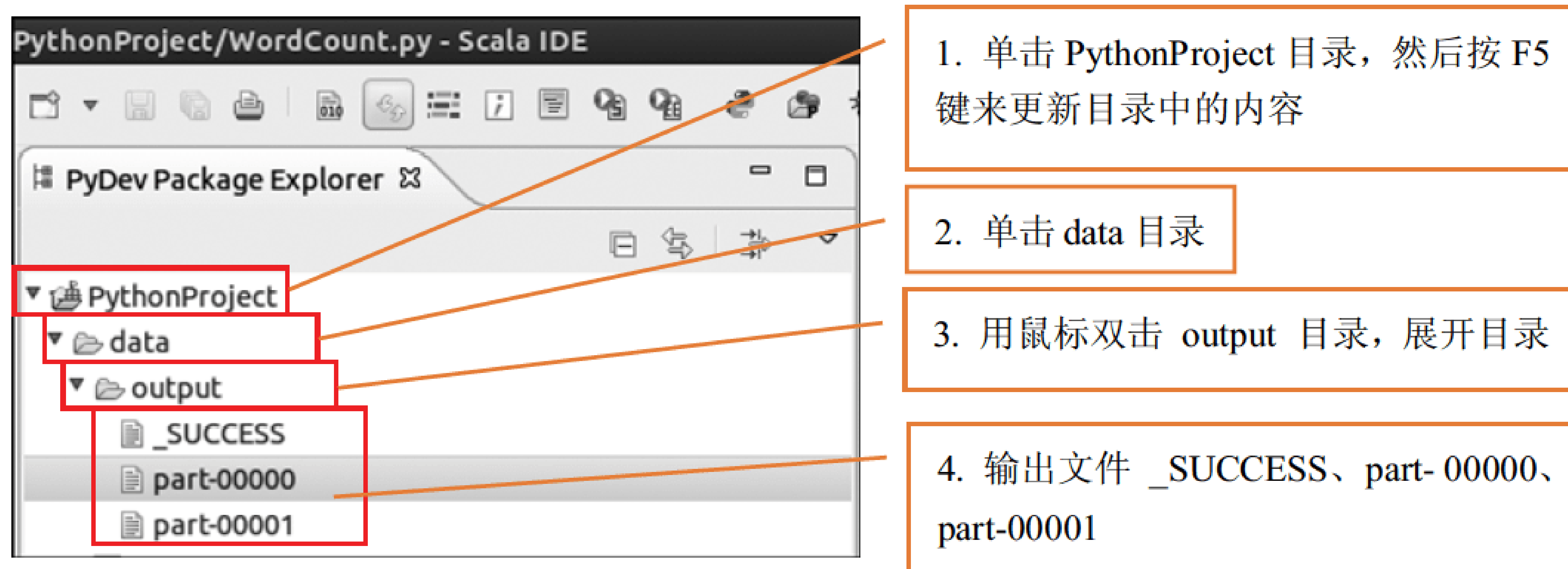


图 11-83　查看输出的 data/output 目录

步骤10 查看输出文件 part-00000

可以用鼠标双击 part-00000 来打开文件，如图 11-84 所示。

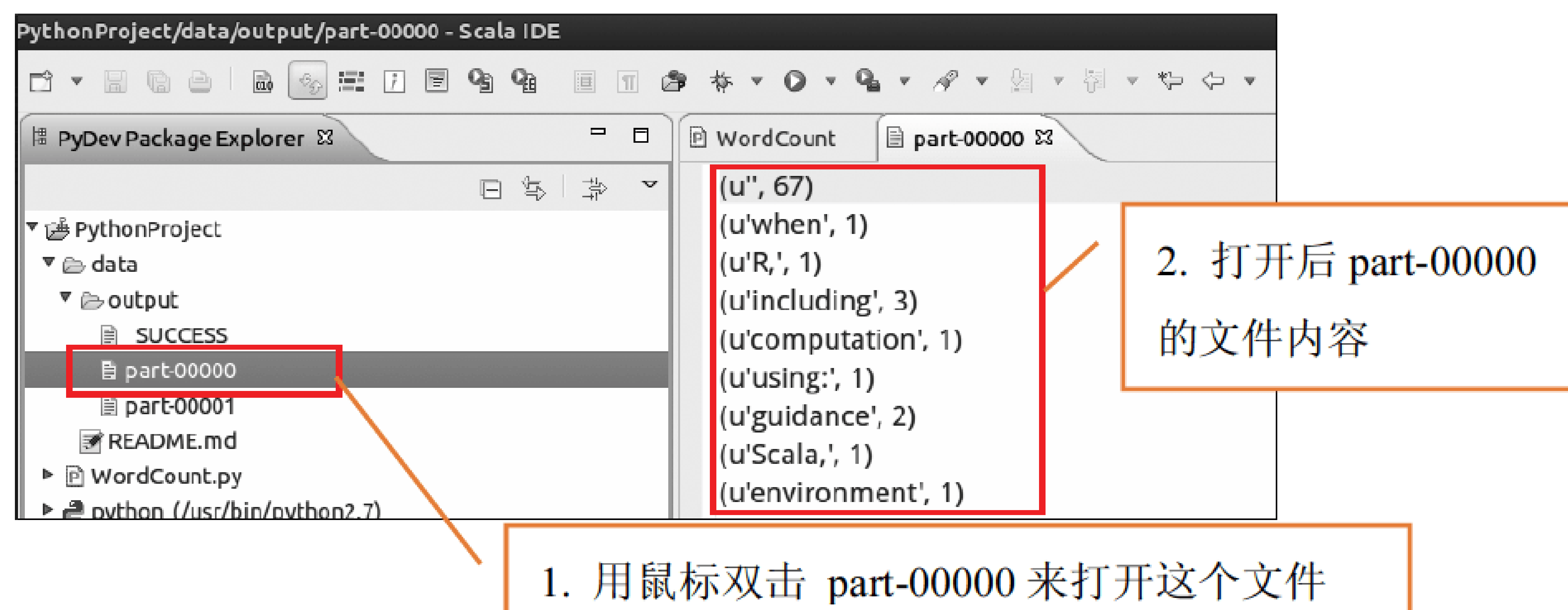


图 11-84　查看输出文件 part-00000

步骤11 再次运行外部工具

之前已经设置完成外部工具，再次运行外部工具的方式如图 11-85 所示。



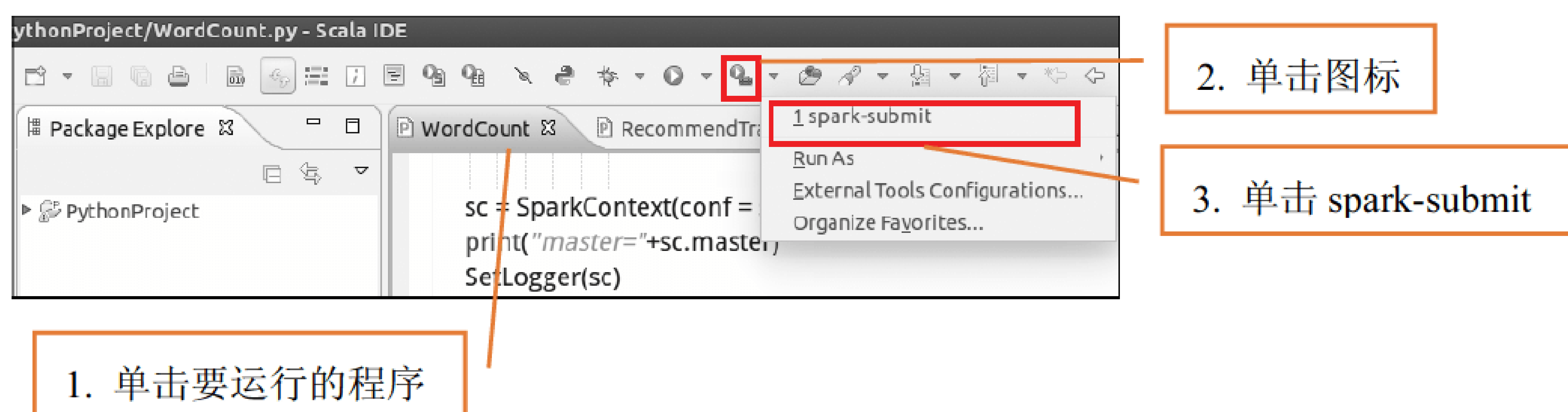


图 11-85　再次运行外部工具

11.16 在 eclipse 运行 spark-submit YARN-client

我们也可以设置外部工具运行在 YARN-client 模式。

步骤 01 设置外部工具（见图 11-86）

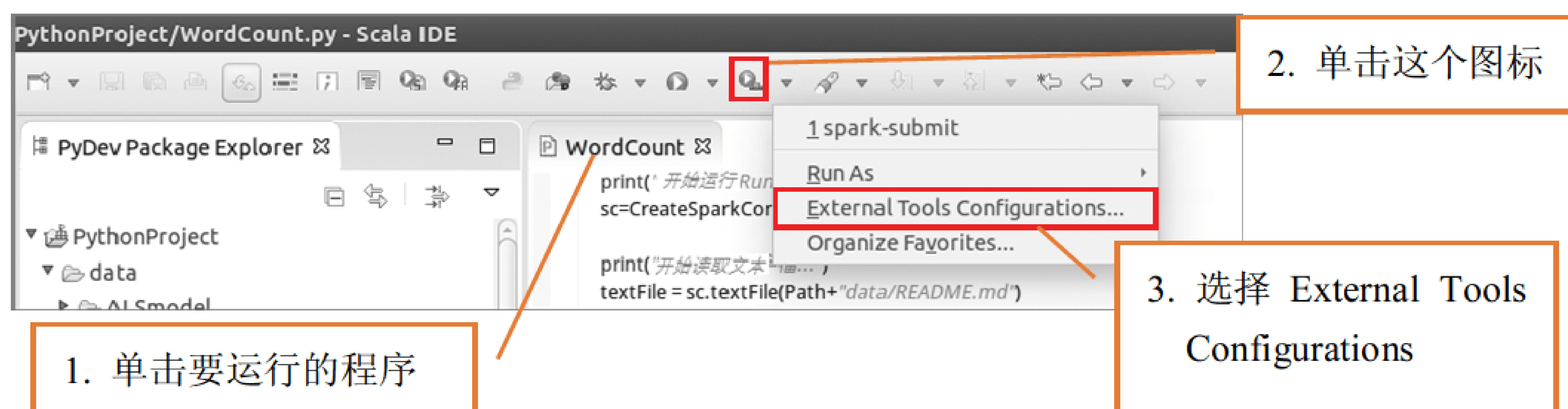


图 11-86　设置外部工具

步骤 02 复制外部工具

我们可以复制之前已经设置完成的 spark-submit 外部工具，如图 11-87 所示。

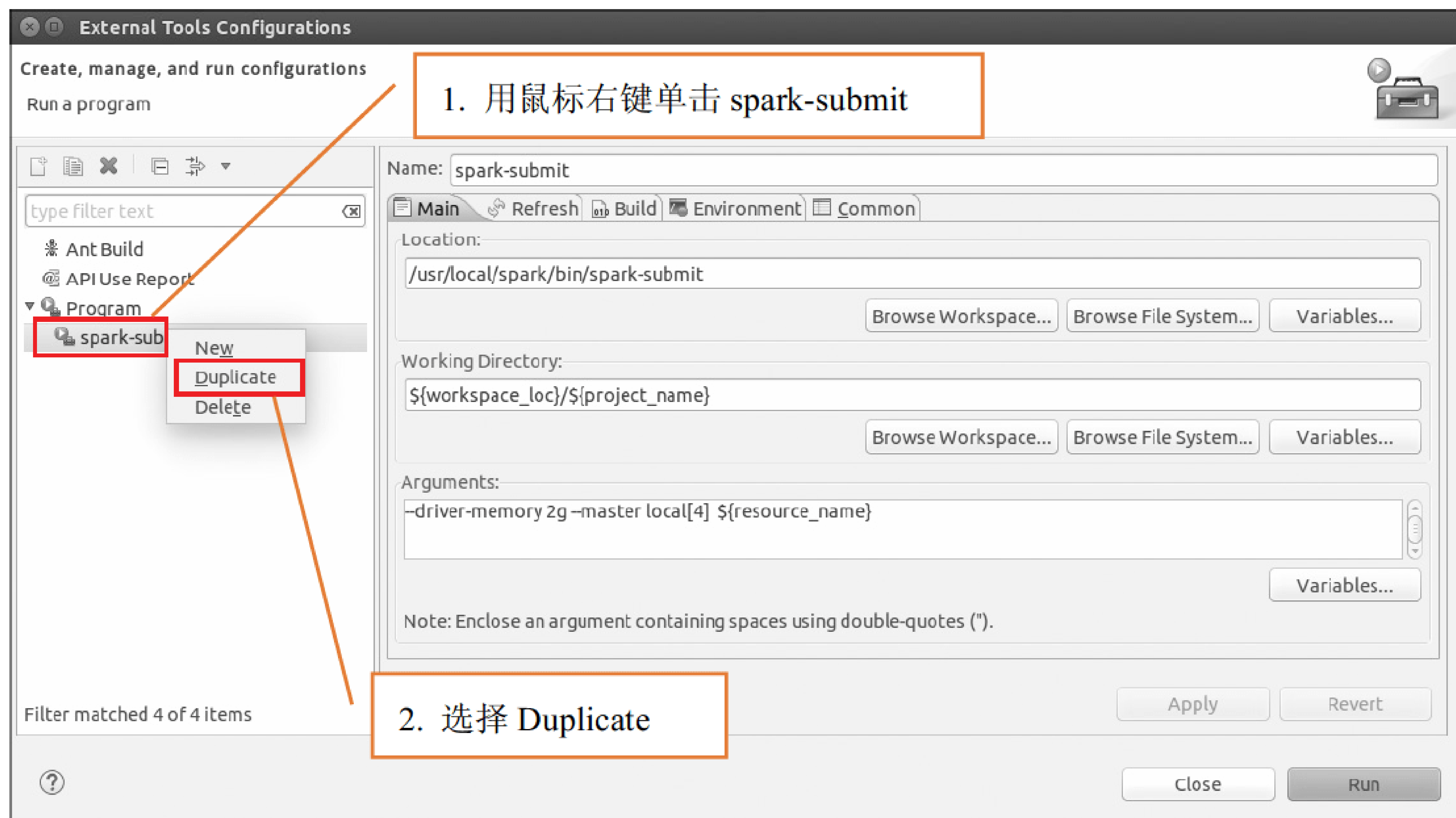


图 11-87　复制

步骤 03 spark-submit YARN-client 设置外部工具（见图 11-88）

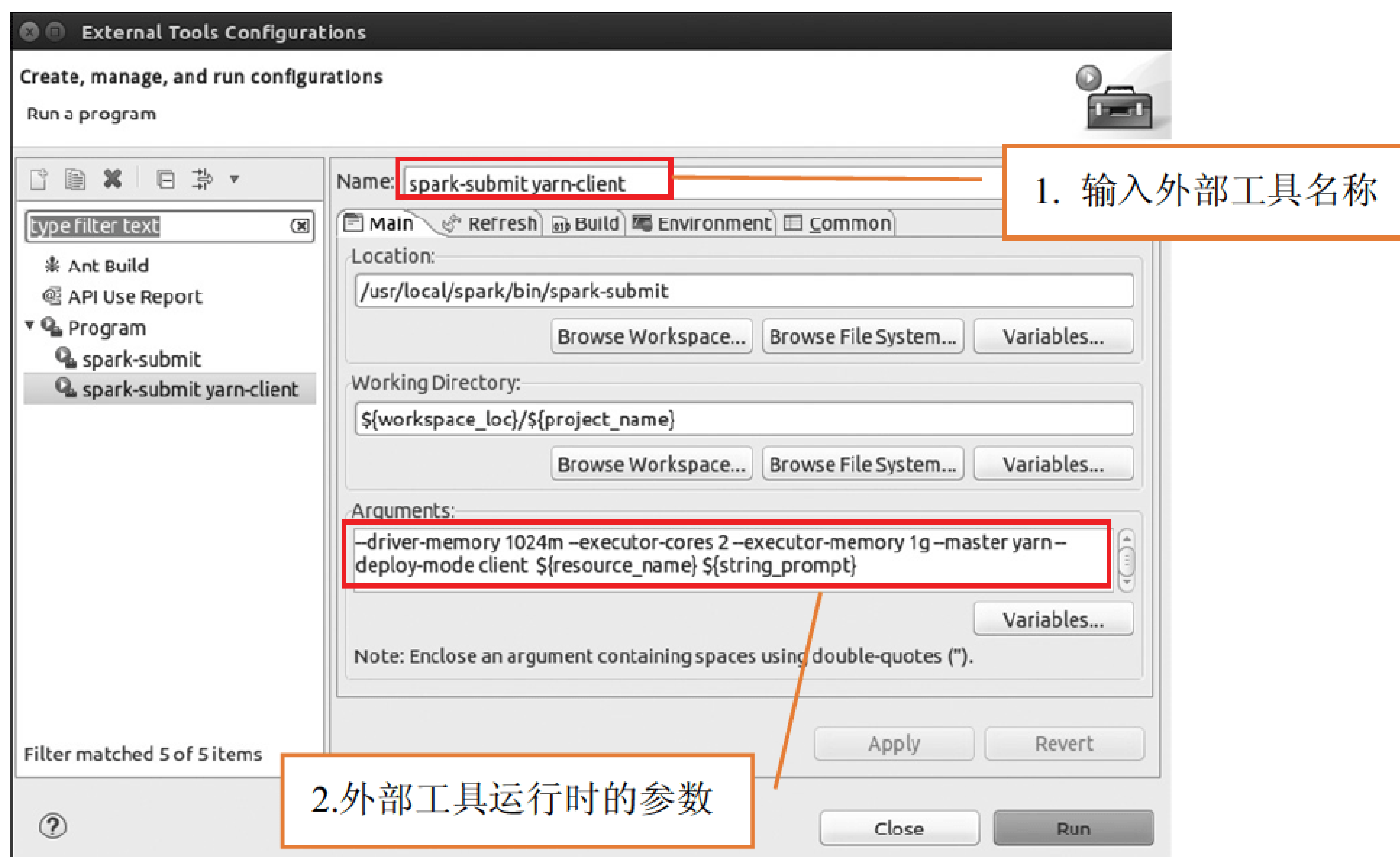


图 11-88　设置外部工具参数

复制后，location 与 Working Directory 完全相同，不需要修改，只需要修改 name 与 Arguments 即可。

➢ name

外部工具的名称会显示在运行外部工具的菜单上。

```
spark-submit yarn-client
```

➢ Arguments

外部工具运行时的参数：

```
--driver-memory 1024m --executor-cores 2 --executor-memory 1g --master yarn
--deploy-mode client ${resource_name} ${string_prompt}
```

其中，${resource_name} 就是当前选中的文件，也就是 WordCount.py，加入${string_prompt}会出现对话框，让用户输入运行参数。

步骤 04 spark-submit YARN-client 设置环境变量（见图 11-89）

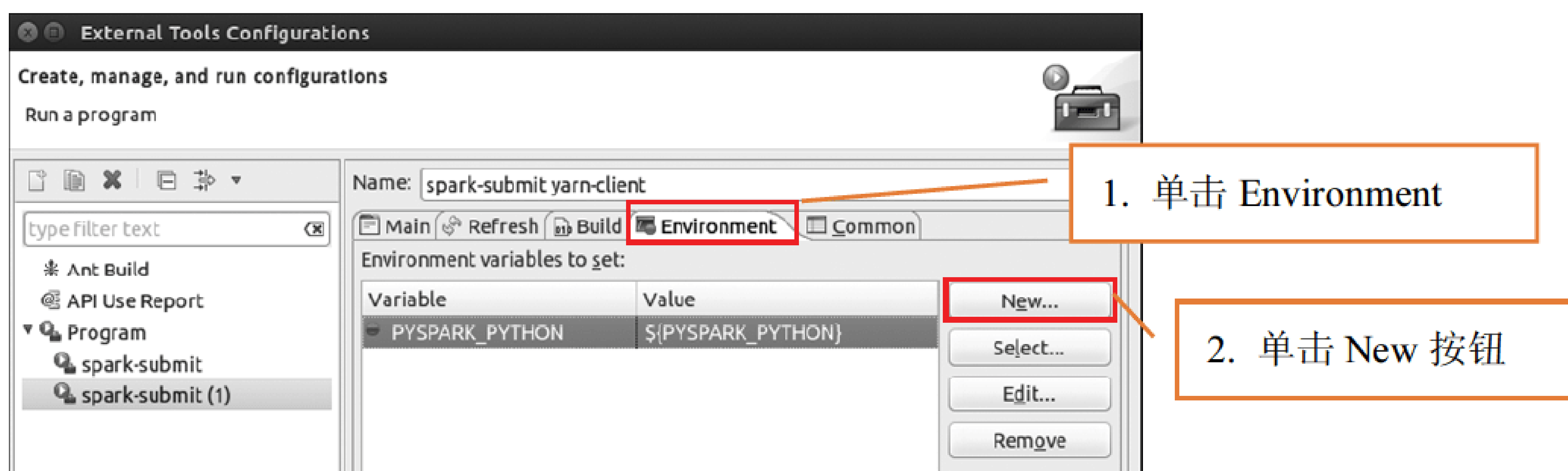


图 11-89　设置环境变量

步骤 05 新建 HADOOP_CONF_DIR 环境变量

因为在 Hadoop YARN 运行，所以必须新建并设置环境变量 HADOOP_CONF_DIR，如图 11-90 所示。



图 11-90　新建 HADOOP_CONF_DIR

步骤 06 运行外部工具（见图 11-91）

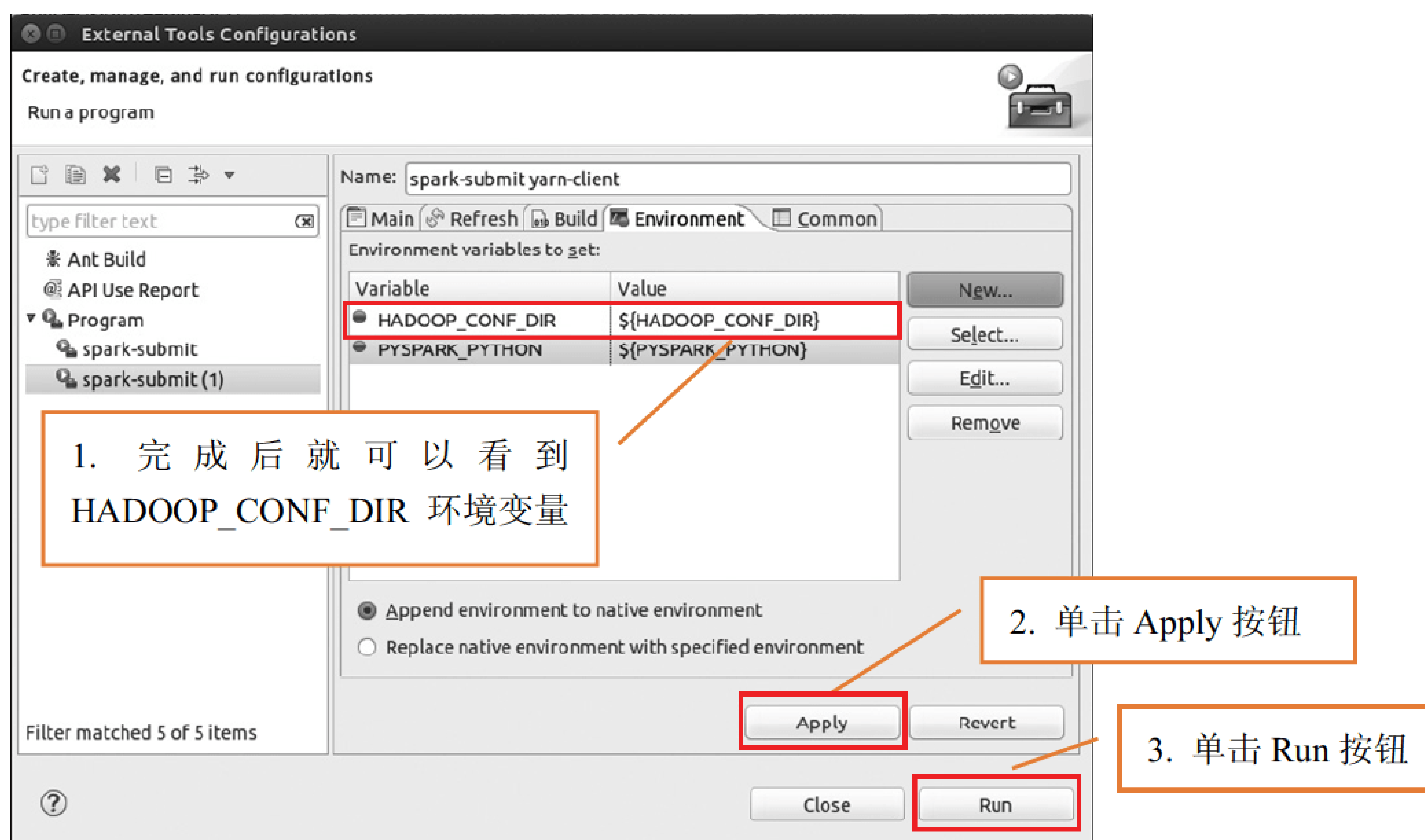


图 11-91 运行外部工具

步骤07 不要输入参数，直接单击 OK 按钮

运行后系统会要求输入程序运行时的参数。由于 WordCount.py 不需要输入参数，因此在这里不要输入参数，直接单击 OK 按钮，如图 11-92 所示。

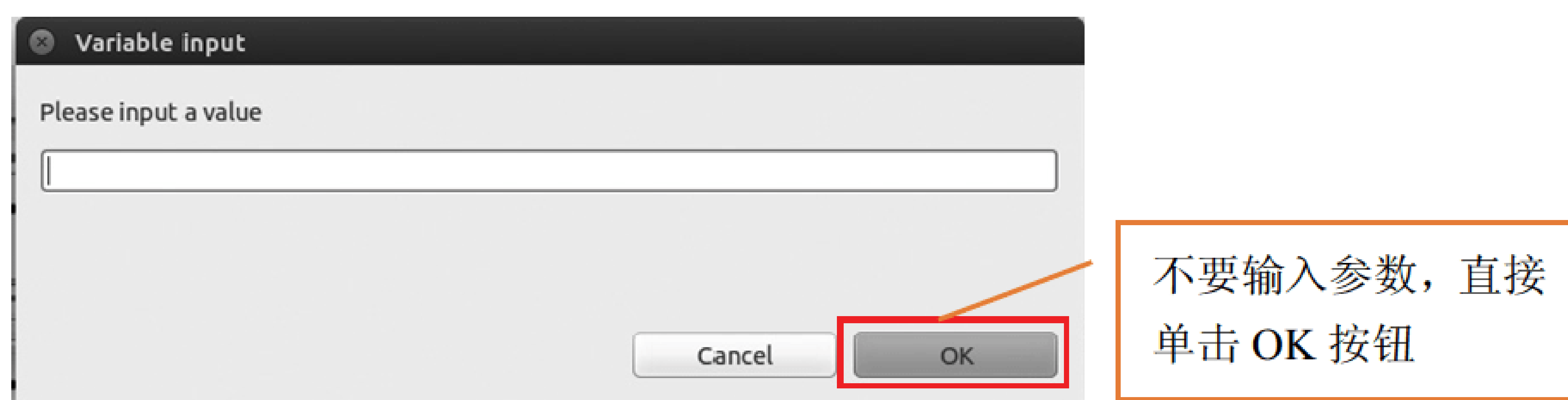


图 11-92 直接单击 OK 按钮

在后续章节中，有些程序运行时是需要输入参数的。

步骤08 spark-submit YARN-client 外部工具运行的结果（见图 11-93）

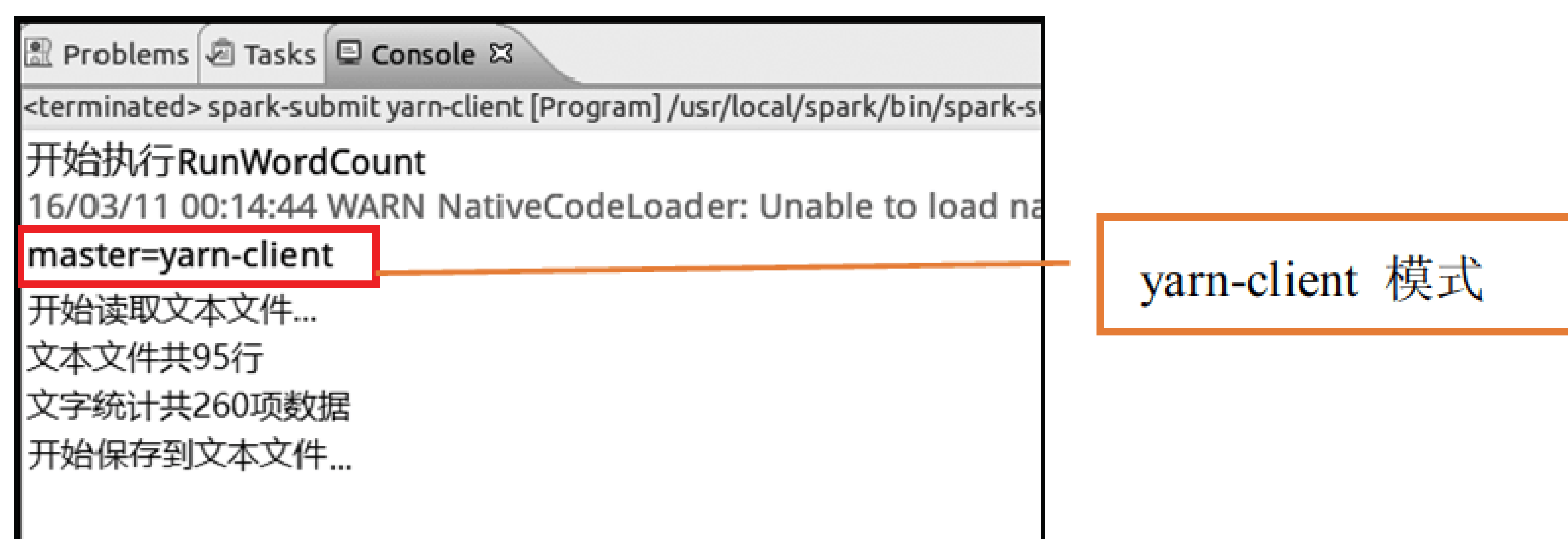


图 11-93 查看外部工具运行的结果



步骤 09 再次运行外部工具

之前已经设置完成外部工具，要再次运行外部工具的方式如图 11-94 所示。

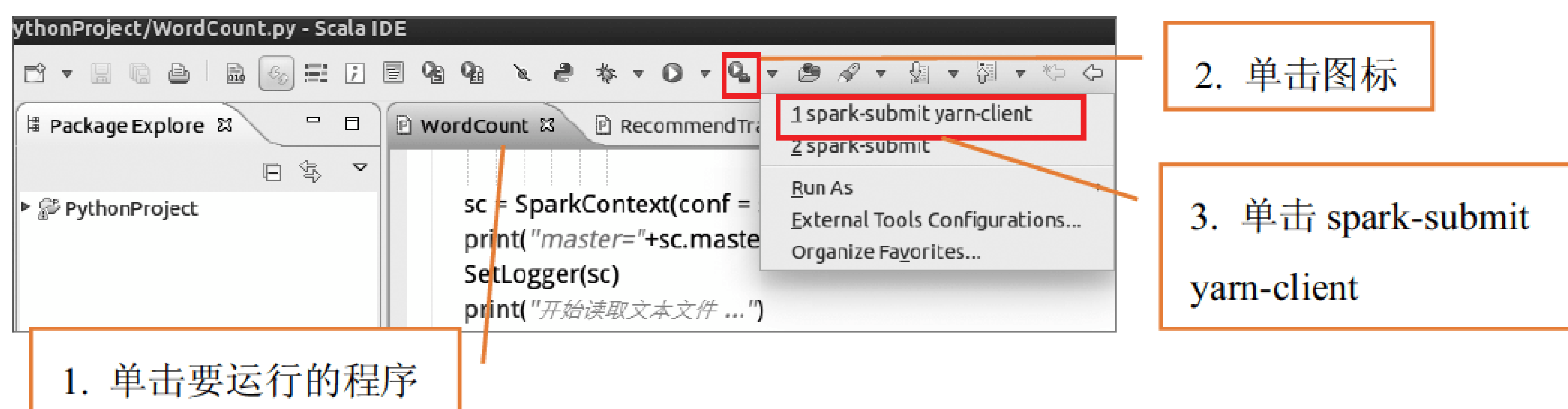


图 11-94　再次运行外部工具

步骤 10 运行外部工具的结果（见图 11-95）

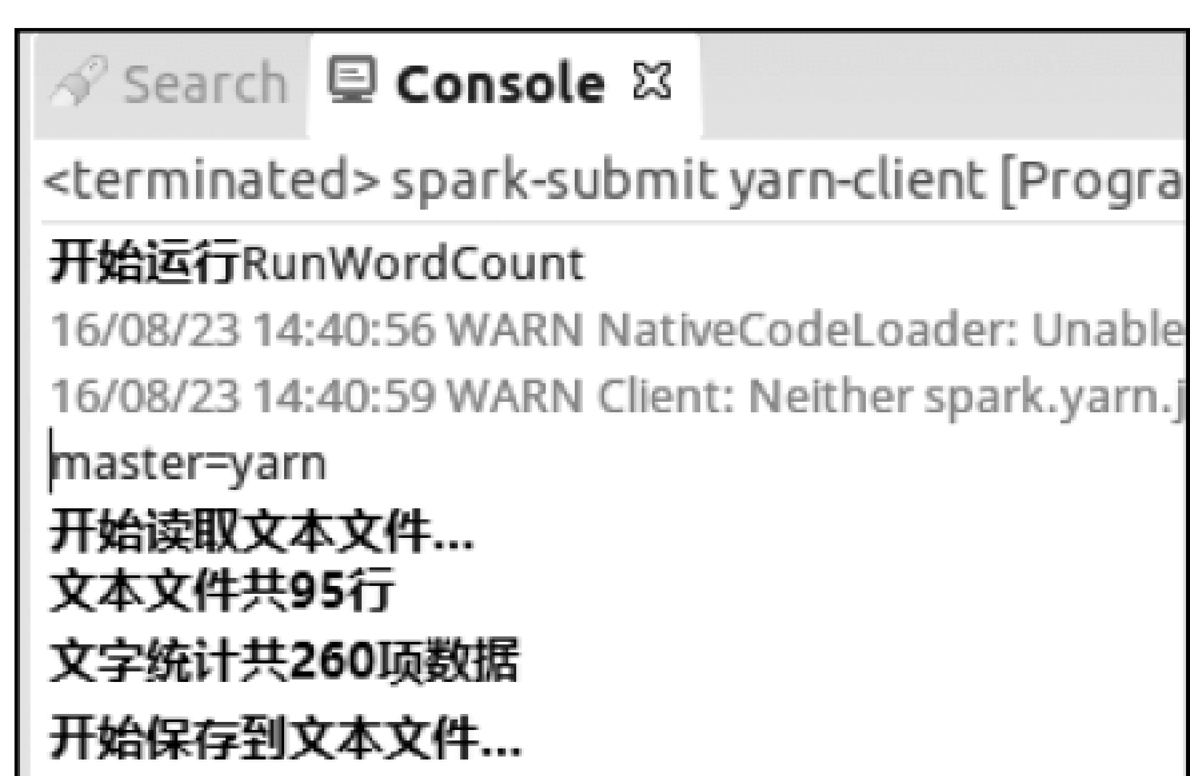


图 11-95　再次运行外部工具的结果

11.17 在 eclipse 运行 spark-submit Standalone

我们也可以设置外部工具运行在 spark-submit Standalone 模式。

步骤 01 设置外部工具（见图 11-96）

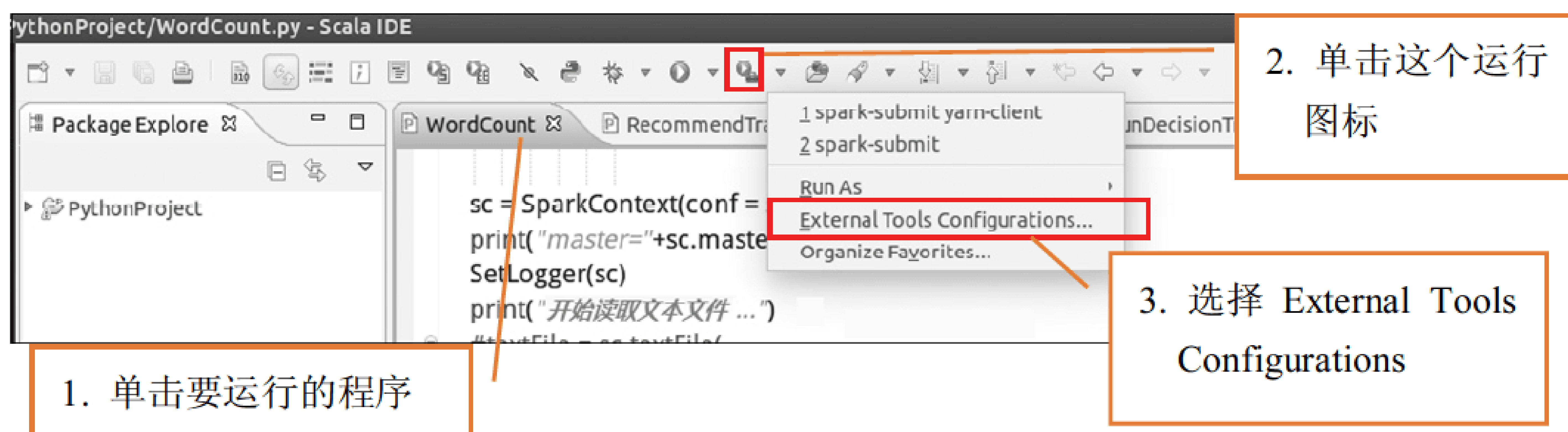


图 11-96　选择设置外部工具菜单

步骤 02 复制外部工具（见图 11-97）

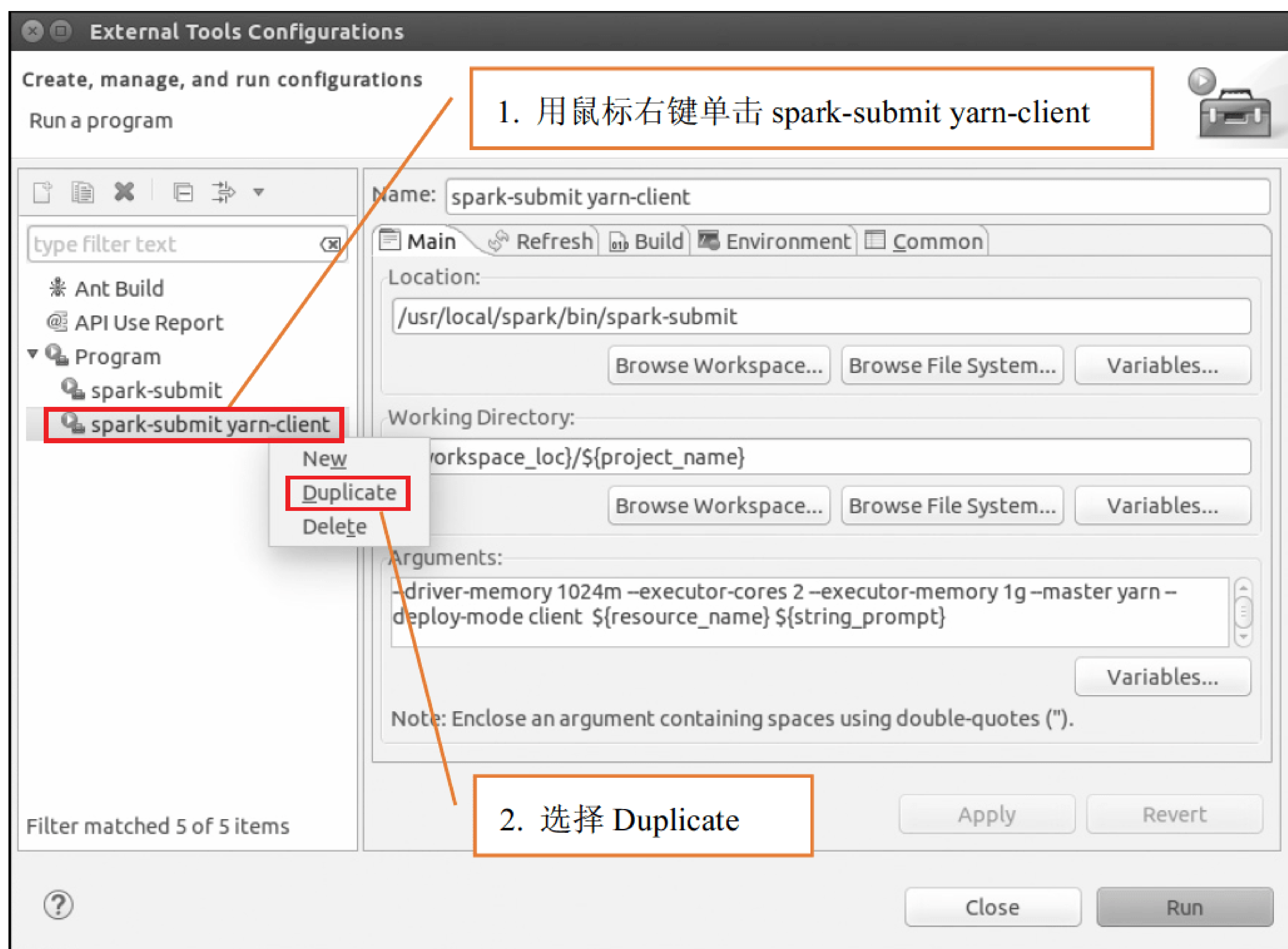


图 11-97 复制外部工具

步骤 03 spark-submit Standalone 设置外部工具（见图 11-98）

复制后，location 与 Working Directory 完全相同，不需要修改，只需修改 Name 与 Arguments 即可。

➢ Name

外部工具的名称会显示在运行外部工具的菜单上。

```
spark-submit Standalone
```

➢ Arguments

外部工具运行时的参数：

```
--master spark://master:7077 --deploy-mode client --executor-memory 500M --total-
executor-cores 2 ${resource_name} ${string_prompt}
```

其中，${resource_name} 就是当前选中的文件，也就是 WordCount.py。



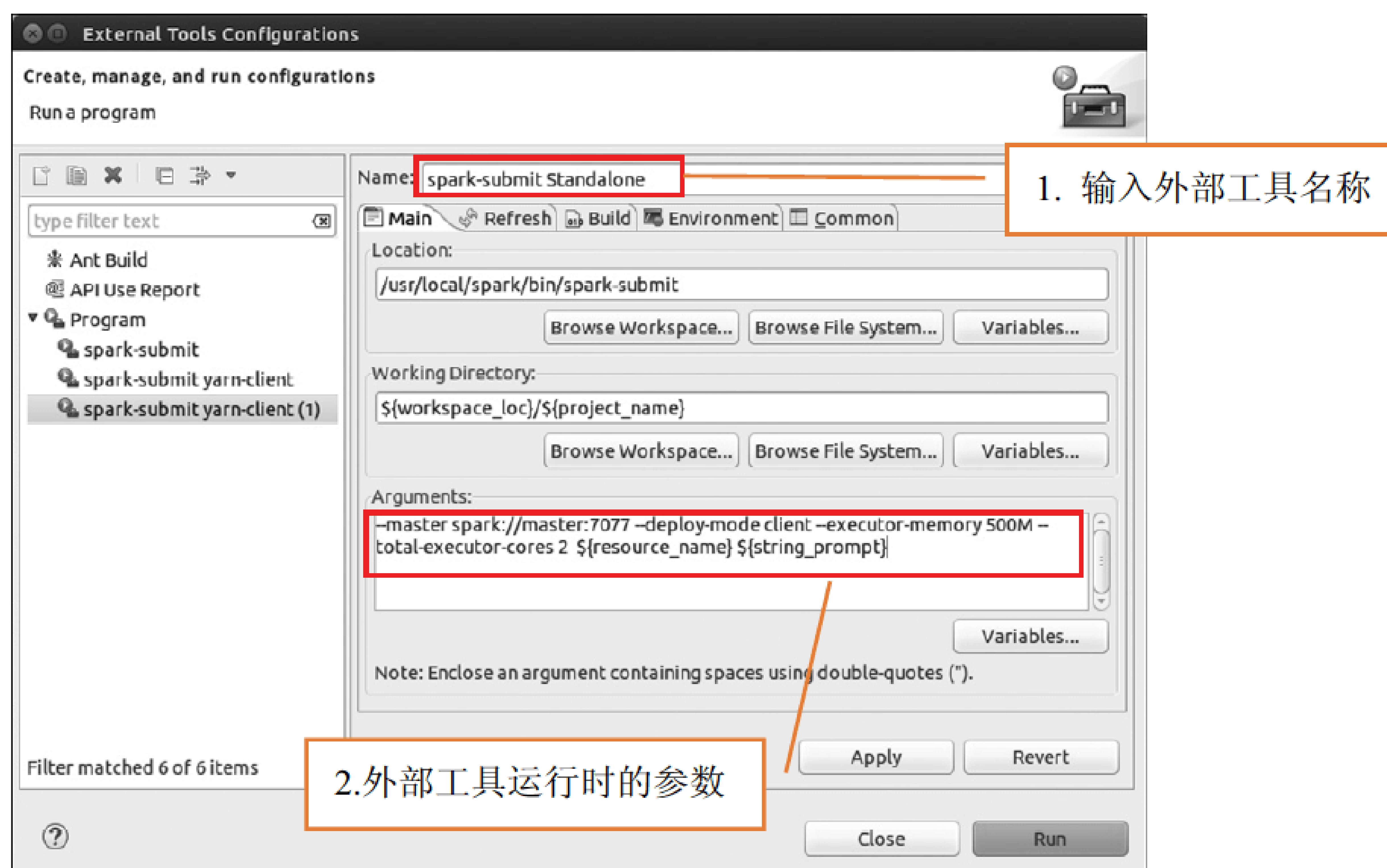


图 11-98　设置外部工具参数

步骤 04 确认环境变量

复制后，环境设置也是相同的，如图 11-99 所示。

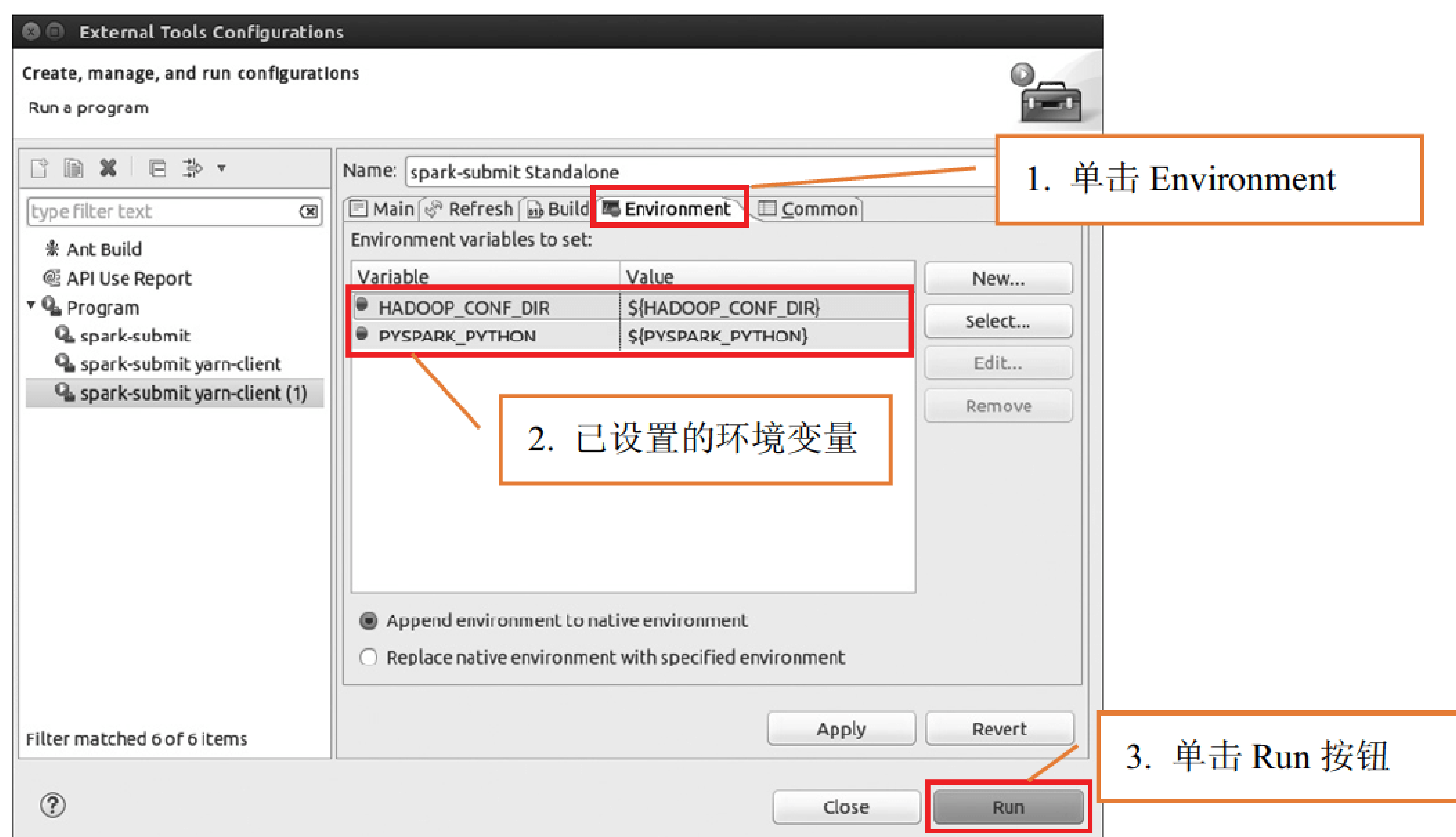


图 11-99　确认环境变量

步骤 05 不要输入参数，直接单击 OK 按钮

运行后系统会要求输入程序运行时的参数。由于 WordCount.py 不需要输入参数，因此在这里不要输入参数，直接单击 OK 按钮，如图 11-100 所示。

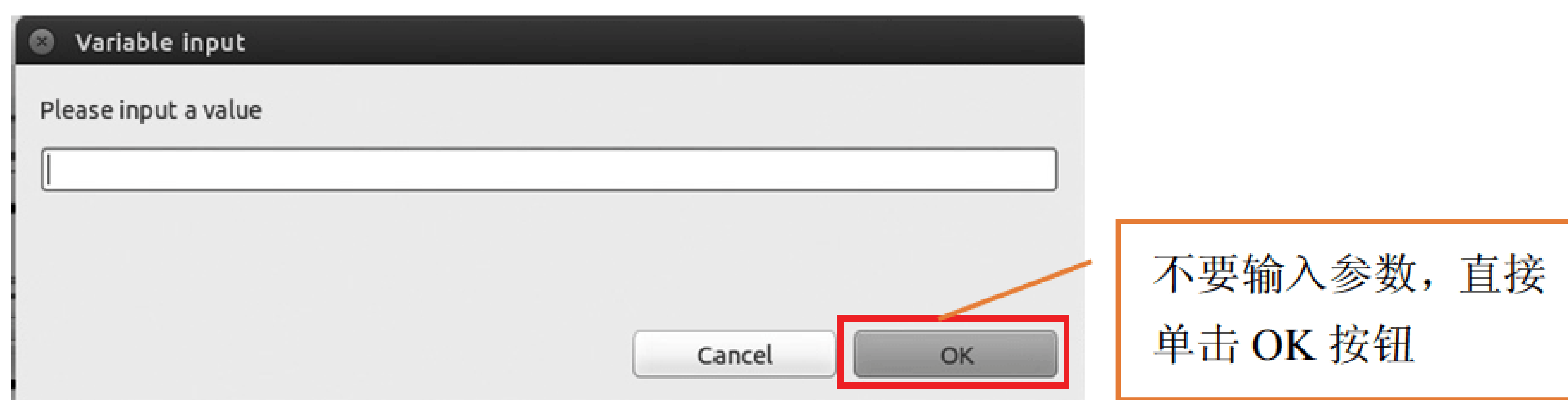


图 11-100　直接单击 OK 按钮

步骤 06 spark-submit spark standalone 运行的结果

在 eclipse 使用外部工具执行 spark-submit spark standalone 的结果，如图 11-101 所示。

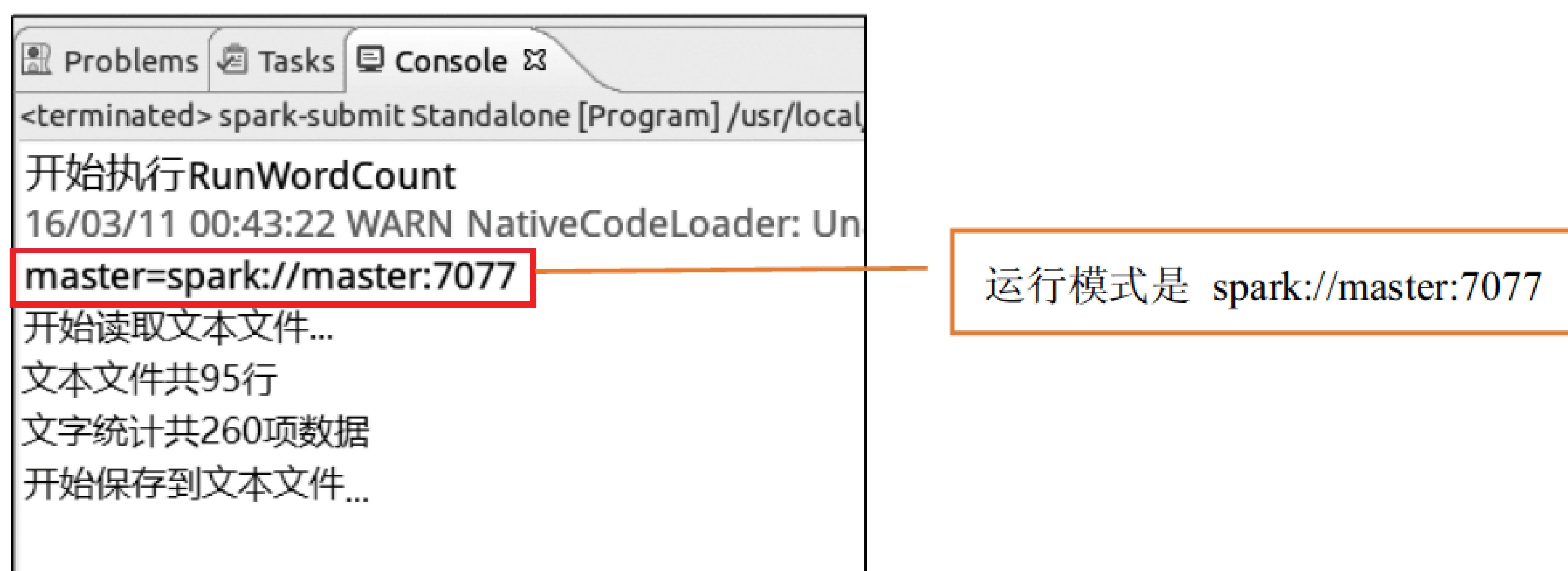


图 11-101　运行结果

步骤 07 再次运行外部工具

如果要再次运行，按照图 11-102 所示的步骤操作即可。

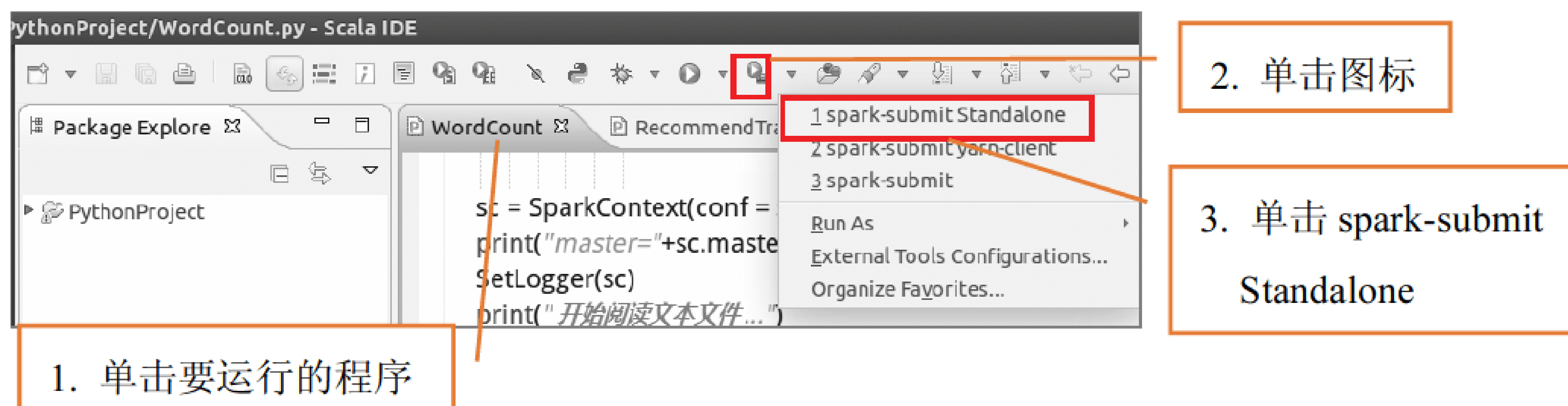


图 11-102　再次运行

11.18 结论

本章我们介绍了如何使用 eclipse 集成开发环境（IDE）来开发 Python Spark 应用程序。接 下来我们将介绍 Spark MLlib 机器学习链接库。

第 12 章

Python Spark创建推荐引擎

推荐引擎是最常见的机器学习应用。我们可以在各大购物网站上看见这方面的应用。

Spark MLlib 支持 ALS（Alternating Least Squares）推荐算法，是机器学习的协同过滤推荐算法。机器学习的协同过滤推荐算法通过观察所有用户给产品的评价来推断每个用户的喜好，并向用户推荐适合的多个产品，也可以把某一个产品推荐给多个用户。

12.1 推荐算法介绍

常见推荐算法如表 12-1 所示。

表 12-1 常见推荐算法

算法	说明
基于关系型规则的推荐（Association Rule）	● 消费者买产品 A，那么他有多大机会购买产品 B ● 购物车分析（啤酒和尿布）
基于内容的推荐（Content-based ）	● 分析网页内容自动分类，再将用户自动分类 ● 将新进已分类的网页推荐给对该群感兴趣的用户
人口统计式的推荐（Demographic）	● 将用户以个人属性（性别、年龄、教育背景、居住地、语言）作为分类的指标 ● 以此类作为推荐的基础
协同过滤式的推荐（Collaborative Filtering）	● 通过观察所有用户对产品的评分来推断用户的喜好 ● 找出对产品评分相近的其他用户，他们喜欢的产品当前用户多半也会喜欢

本章主要介绍协同过滤式的推荐，优缺点如表 12-2 所示。

表 12-2 协同过滤式推荐的优缺点

优点	缺点
• 可以达到个性化推荐 • 不需要内容分析 • 可以发现用户新的兴趣点 • 自动化程度高	• 冷启动问题（Cold-start）：如果没有历史数据就没办法分析 • 新用户问题：新用户没有评分，就不知道他的喜好

具体实现上多半是使用混合式方法。结合上述方法可以达到互补效果，进而提供给用户更好的个性化推荐体验。

12.2 “推荐引擎”大数据分析使用场景

假设有一个 MoviesOnLine 的在线电影网站，会员可以付费在线观赏电影。公司希望能运用大数据分析推荐引擎增加会员观看影片的次数，以增加营收，因此找来了大数据分析师并组成了一个团队（负责“电影推荐引擎”大数据项目）。

➢ **找出问题**

“问对问题”是解决问题的第一步。大数据分析师与公司人员讨论后发现，现有公司的推

荐系统是人口统计式的推荐（Demographic），必须具有个人属性数据。基于隐私权的原因，越来越难搜集到正确的个人属性数据，而且相同属性的用户未必会有相同的喜好，无法做到个性化的推荐。

➢ **设计解决方案模型**

大数据分析师与公司人员讨论后，决定将推荐引擎加入协同过滤式的推荐（Collaborative Filtering）。这种方法是通过观察所有会员给影片的评分来推断每个会员的喜好，并向会员推荐适合的影片。也就是说，和你观看影片喜好相近的用户，他喜欢的电影你多半也会喜欢。通过这种方式，可以达到个性化的推荐效果。

➢ **搜集数据**

协同过滤式的推荐（Collaborative Filtering）最大的缺点是冷启动问题（Cold-start），如果没有历史数据就没办法分析推荐。网站已经运营了一段时间，累积了很多会员电影评分的数据和观看记录，所以可以使用这些数据构建协同过滤式的推荐引擎。

➢ **创建模型**

大数据分析师决定使用 Spark MLlib 的 ALS（Alternating Least Squares）推荐算法。采用这种方式可以解决稀疏矩阵（Sparse Matrix）的问题。即使是大量的用户与产品，也能够在合理的时间内完成运算。在使用历史数据训练后，就可以创建模型。

➢ **进行推荐**

有了模型之后，就可以使用模型进行推荐了。设计如下推荐功能，可以增加会员观看电影的次数：

- **针对用户推荐感兴趣的电影**：可以针对每一个会员，定期发送短信或 E-mail；或会员登录时，推荐给他/她可能会感兴趣的电影。
- **针对电影推荐给感兴趣的用户**：当想要促销某些电影时，可以找出可能会对这些电影感兴趣的会员，并且发送短信或 E-mail。

12.3 ALS 推荐算法的介绍

Spark MLlib 支持的 ALS 推荐算法是机器学习的协同过滤式推荐算法。机器学习的协同过滤式推荐算法通过观察所有用户给产品的评分来推断每个用户的喜好，并向用户推荐适合的产品。

步骤 01 用户对产品项目的评分

用户对产品项目的评分有两种方式。现以只有 5 个用户与 5 个项目为例，说明如下：

➢ **显式评分（Explicit Rating）**

网站上的设计经常会请用户对某个产品进行评分，例如评分 1~5 颗星。可将其整理为下列矩阵：

	Item A	Item B	Item C	Item D	Item E
User 1	2	1	5		
User 2	1	3	1	1	
User 3	3			4	
User 4	2		2	1	2
User 5	1	1	1	4	1

➢ **隐式评分（Implicit rating）**

有时在网站的设计上并不会请用户对某个产品进行评分，但是会记录用户是否选择了某个产品。如果选择了某个产品，就代表用户可能对该产品感兴趣，但是我们不知道评分为几颗星，这种方式称为隐式评分（1 代表用户对该项产品有兴趣）。

	Item A	Item B	Item C	Item D	Item E
User 1	1	1	1		
User 2	1	1	1	1	
User 3	1			1	
User 4	1		1	1	1
User 5	1	1	1	1	1

推荐算法就是找出两个用户喜好的相似性。

例如，User 1 有兴趣的项目为(A、B、C)，User 2 有兴趣的项目为(A、B、C、D)。User1 只比 User2 少了一个喜好项目 D。因此，当推荐算法要推荐项目给 User 1 时就会推荐项目 D。

步骤 02 稀疏矩阵（Sparse Matrix）的问题

如图 12-1 所示，当用户与项目评分越来越多时，会发现大部分都是空白的，这称为稀疏矩阵。

用户与产品项目评分成千上万时，整个矩阵就会很大，而且很多都是空白。使用计算机来处理这样的矩阵很浪费内存，而且处理需要花费很多时间。

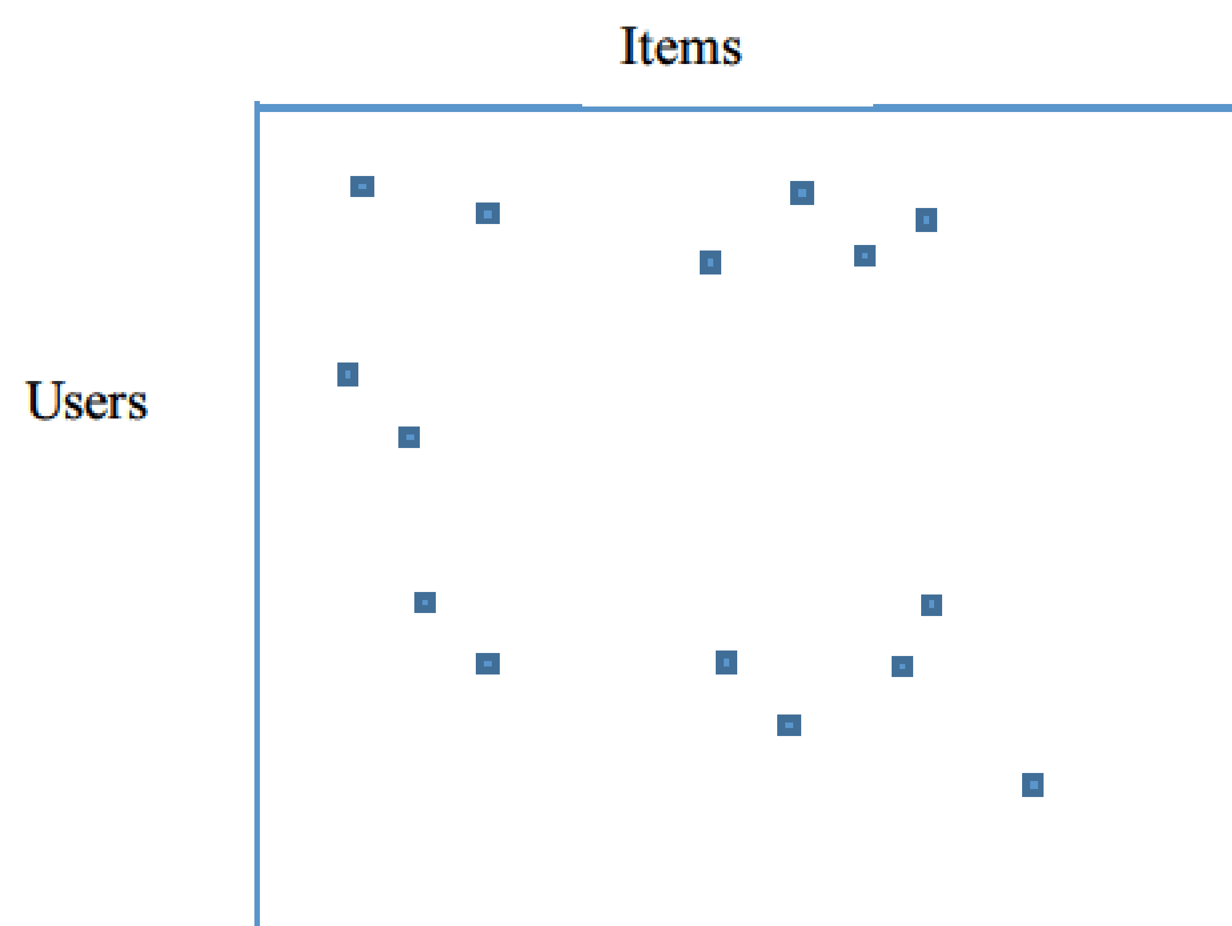


图 12-1　稀疏矩阵示意图

步骤 03 矩阵分解（Matrix Factorization）

要解决稀疏矩阵的问题，需采用矩阵分解。

如图 12-2 所示，将原本矩阵 $\boldsymbol{A}(m\times n)$ 分解成 $\boldsymbol{X}(m\times \mathrm{rank})$ 矩阵与 $\boldsymbol{Y}(\mathrm{rank}\times n)$ 矩阵，而且 $\boldsymbol{A}$ 大约等于 $\approx\boldsymbol{X}\times\boldsymbol{Y}$。

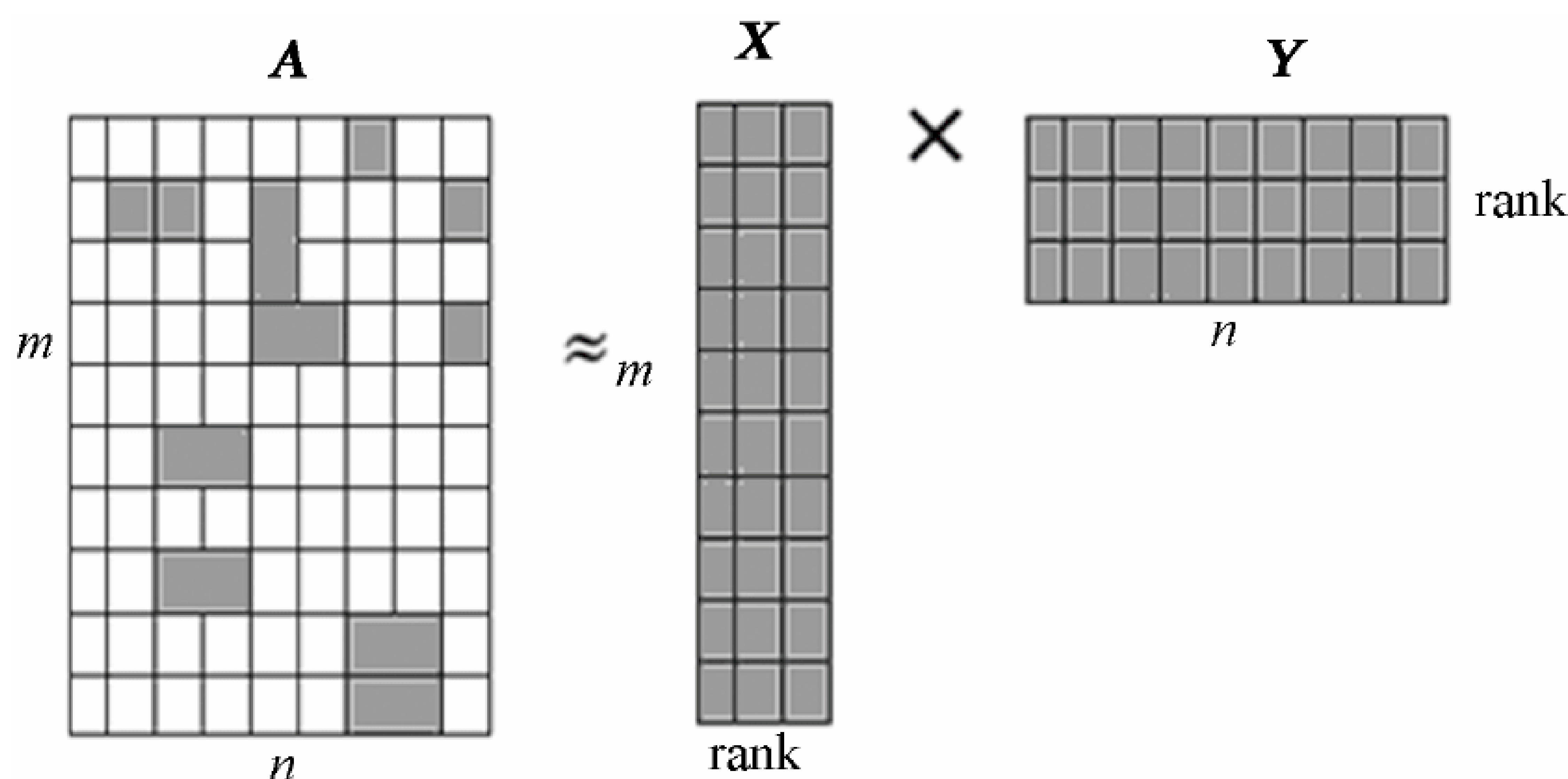


图 12-2　矩阵分解

12.4 如何搜索数据

MovieLens 是一个推荐系统和虚拟社区网站，主要功能是使用协同过滤技术向会员推荐电影。GroupLens Research 实验室隶属于明尼苏达大学，MovieLens 网站只是其中一个项目。在浏览器输入下列网址进入 Movielens 网站：

http://grouplens.org/datasets/movielens/

在下列网页中（见图 12-3），可以看到不同大小的 MovieLens 数据集。考虑到大部分读者是使用个人计算机进行练习，所以在本书中采用 ml-100k 作为范例数据集。如果具有比较强大的服务器，也可以下载更大的数据集来练习。

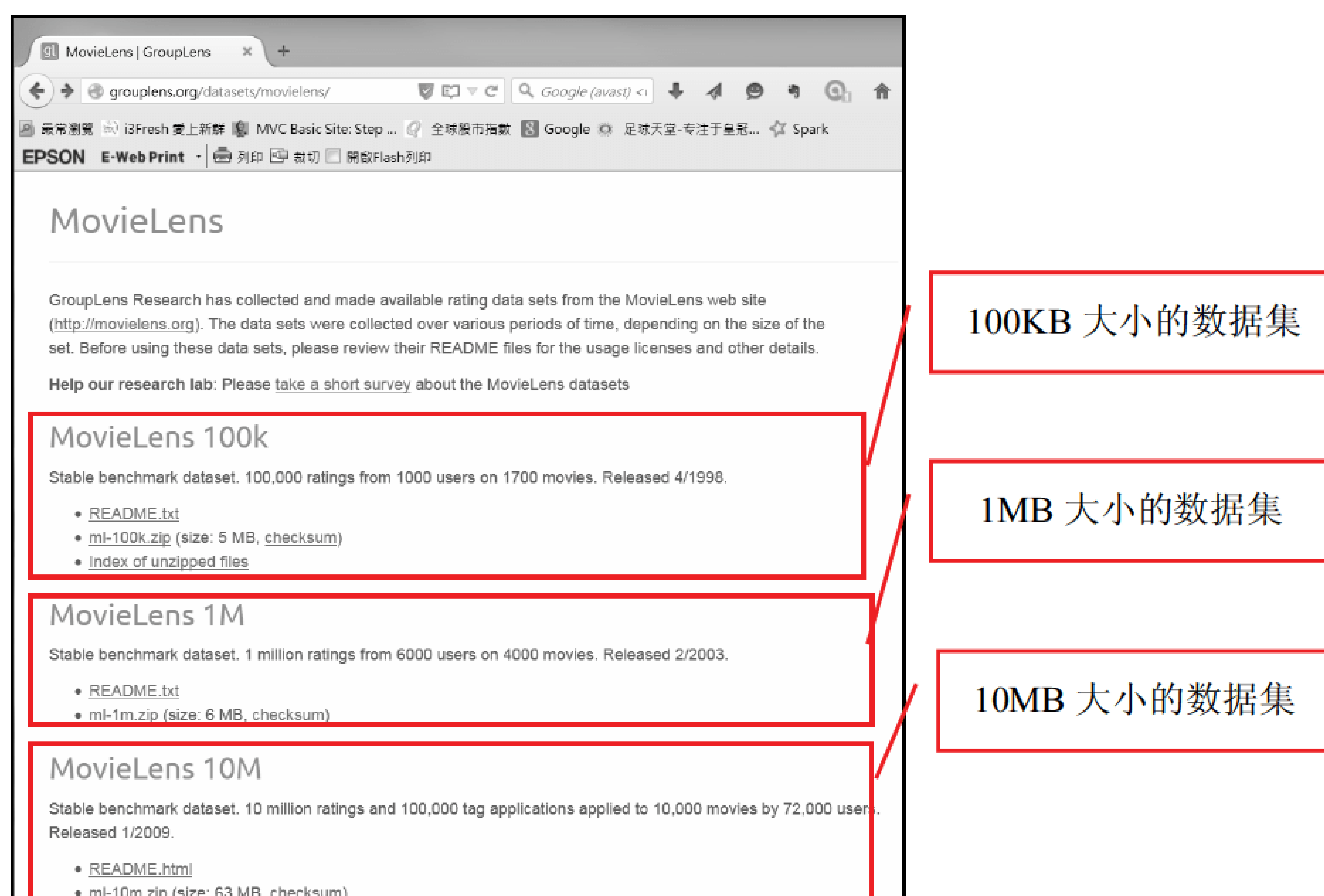


图 12-3　根据计算机的计算能力选择数据集

步骤 01　下载 ml-100k 数据

在“终端”程序中输入下列命令，下载 ml-100k 数据。

➢ **创建存储下载数据的目录**

创建默认存储下载数据的目录，加上“-p”选项会创建多层目录。

```
mkdir -p ~/pythonwork/PythonProject/data
```

➢ **切换到默认存储数据的目录**

```
cd ~/pythonwork/PythonProject/data
```

➢ **下载 ml-100k.zip**

```
wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
```

运行后屏幕显示界面如图 12-4 所示。

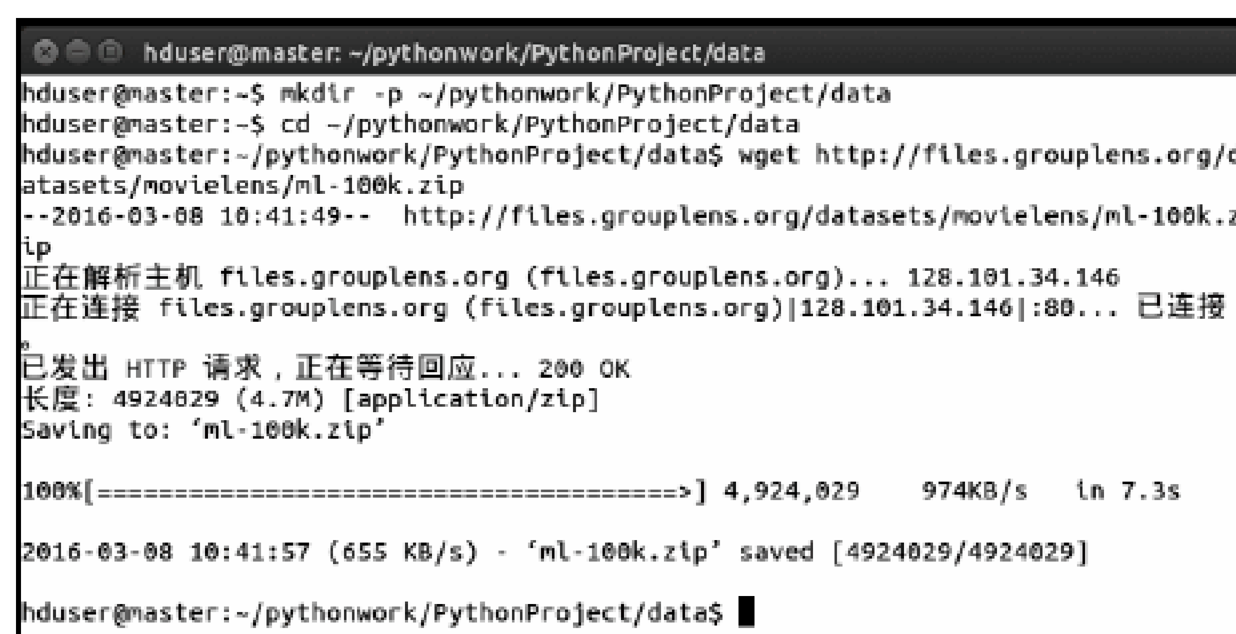


图 12-4　下载 ml-look 数据

步骤 02　解压缩 ml-100k 数据

在“终端”程序中输入下列命令，解压缩 ml-100k.zip 文件。

➢ **解压缩 ml-100k**

在指令中加入“-j”选项可使文件解压缩到当前目录。

```
unzip -j ml-100k
```

运行后屏幕显示界面如图 12-5 所示。

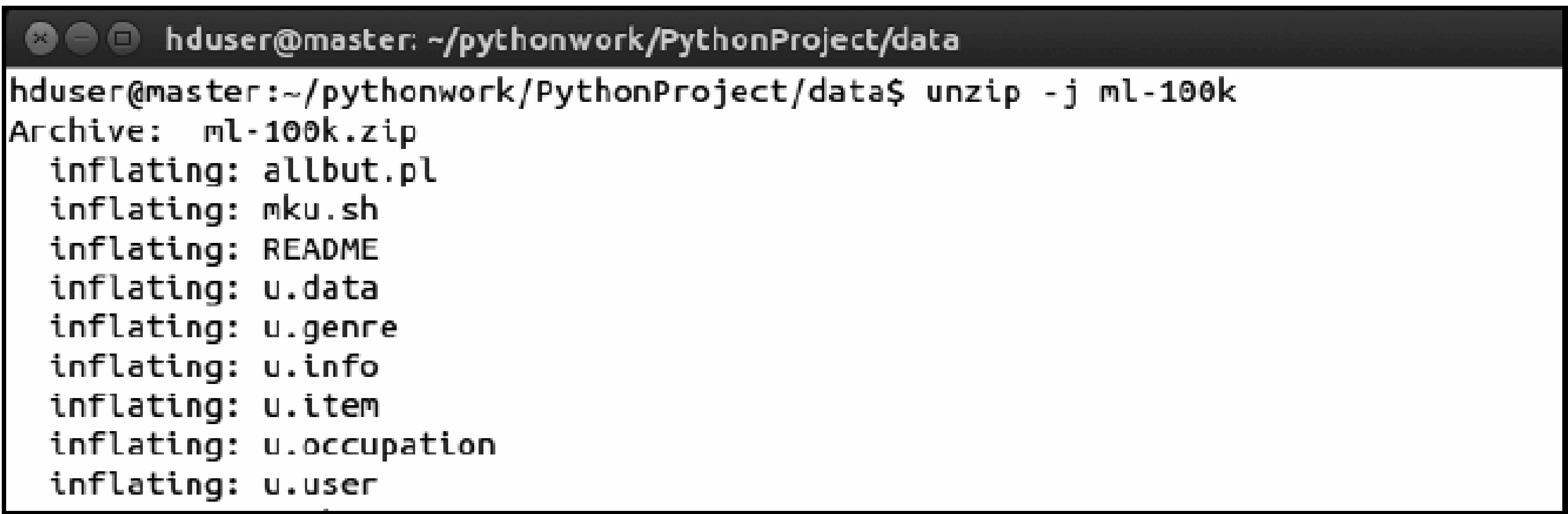


图 12-5　解压缩 ml-100k 数据

步骤 03　ml-100k 文件数据的说明

可以在“终端”程序中使用 ll 命令查看有哪些文件，本章主要会使用到表 12-3 所示的两个数据文件。

表 12-3　数据文件

数据文件名称	说明
u.dat 用户评分数据	包含 4 个字段：user id（用户 id）、item id（项目 id）、rating（评分）、timestamp（日期时间）
u.item 电影的数据	具有很多个字段，本书主要使用前两个字段：movie id（电影 id）、movie title（电影片名）

步骤 04　启动 Hadoop Multi Node Cluster

启动我们之前所创建的 Hadoop Multi Node Cluster 中所有的虚拟机，如图 12-6 所示。



图 12-6　启动虚拟机

步骤05 启动 Hadoop Cluster

在 master 服务器的“终端”程序中输入下列命令：

➢ **启动 Hadoop 群集 Cluster**

```
start-all.sh
```

运行后屏幕显示界面如图 12-7 所示。

```
hduser@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-master.out
data1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data1.out
data2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data2.out
data3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-data3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-master.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-master.out
data1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data1.out
data2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data2.out
data3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-data3.out
```

图 12-7 启动 Hadoop Cluster

步骤06 复制 ml-100k 文件到 HDFS 中

在“终端”程序中输入下列命令，复制 ml-100k 文件到 HDFS 中。

➢ **创建 HDFS 目录**

```
hadoop fs -mkdir /user/hduser/data
```

➢ **复制文件**

```
hadoop fs -copyFromLocal -f ~/pythonwork/PythonProject/data /user/hduser/
```

➢ **查看 HDFS 测试文件**

```
hadoop fs -ls /user/hduser/data
```

运行后屏幕显示界面如图 12-8 所示。

```
hduser@master:~/pythonwork/PythonProject/data$
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -mkdir /user/hduser/data
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -copyFromLocal -f ~/pythonwork/PythonProject/data  /user/hduser/
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -ls /user/hduser/data
Found 26 items
-rw-r--r--   3 hduser supergroup       6750 2016-06-26 21:55 /user/hduser/data/README
-rw-r--r--   3 hduser supergroup       3359 2016-06-26 21:55 /user/hduser/data/README.md
-rw-r--r--   3 hduser supergroup        716 2016-06-26 21:55 /user/hduser/data/allbut.pl
-rw-r--r--   3 hduser supergroup        643 2016-06-26 21:55 /user/hduser/data/mku.sh
-rw-r--r--   3 hduser supergroup    4924029 2016-06-26 21:55 /user/hduser/data/ml-100k.zip
drwxr-xr-x   - hduser supergroup          0 2016-06-26 21:55 /user/hduser/data/output
-rw-r--r--   3 hduser supergroup    1979173 2016-06-26 21:55 /user/hduser/data/u.data
-rw-r--r--   3 hduser supergroup        202 2016-06-26 21:55 /user/hduser/data/u.genre
-rw-r--r--   3 hduser supergroup         36 2016-06-26 21:55 /user/hduser/data/u.info
-rw-r--r--   3 hduser supergroup     236344 2016-06-26 21:55 /user/hduser/data/u.item
-rw-r--r--   3 hduser supergroup        193 2016-06-26 21:55 /user/hduser/data/u.occupation
-rw-r--r--   3 hduser supergroup      22628 2016-06-26 21:55 /user/hduser/data/u.user
```

图 12-8 复制 ml-look 文件到 HDFS 中



12.5 启动 IPython Notebook

为了让大家更容易理解 ALS 算法，我们使用 IPython Notebook 来进行示范（IPython Notebook 具有互动性的好处）。以下示范在 Hadoop YARN-client 模式运行，也可以参考第 9.9 节的说明在不同的模式运行 IPython Notebook。

步骤 01 启动 IPython Notebook 在 Hadoop YARN-client 模式

在“终端”程序中输入下列命令：

➢ **启动 Hadoop Cluster**

参考第 8.6 节，先启动 master、data1、data2、data3，然后执行 start-all.sh。

```
Start-all.sh
```

➢ **切换到 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **在 Hadoop YARN-client 模式运行 IPython Notebook**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" HADOOP_CONF_
DIR=/usr/local/hadoop/etc/hadoop pyspark --master yarn --deploy-mode client
```

按 Enter 键后，就会启动浏览器，显示 IPython Notebook 的界面，如图 12-9 所示。

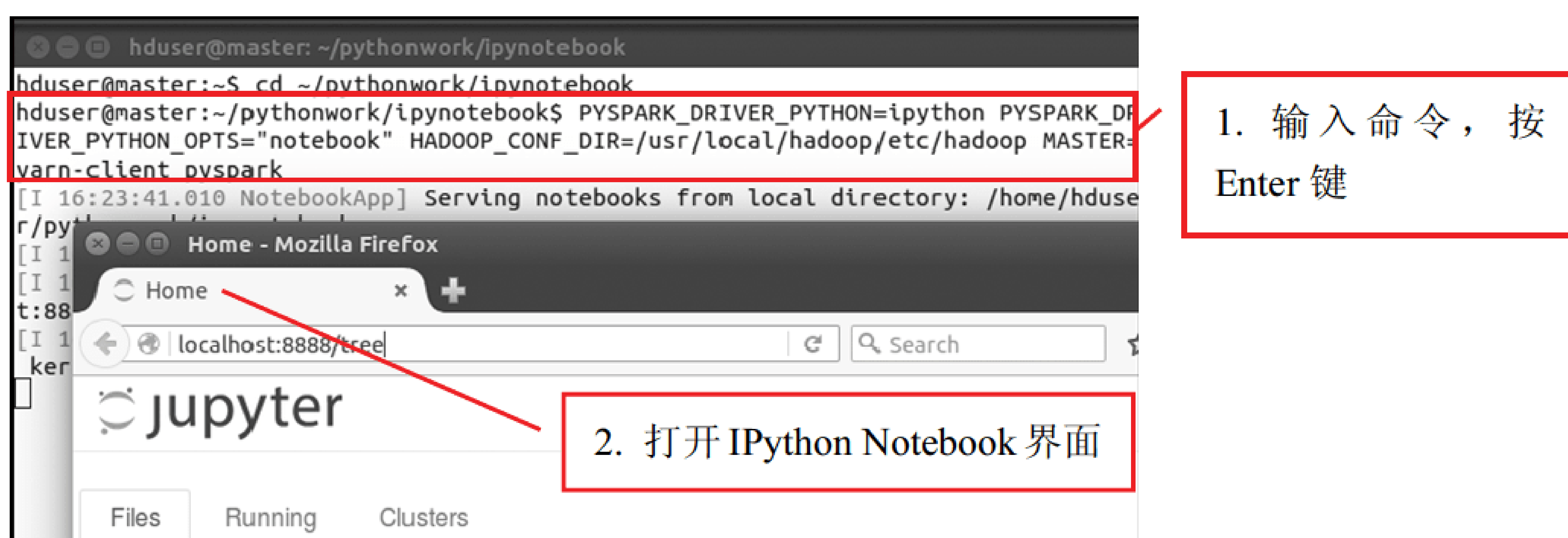


图 12-9　打开 IPython Notebook 界面

步骤 02 新建一个 IPython Notebook

进入 IPython Notebook 界面，可以按照图 12-10 所示的步骤新建 NoteBook。



图 12-10　新建 IPython Notebook

步骤 03 已新建好的 IPython Notebook（见图 12-11）

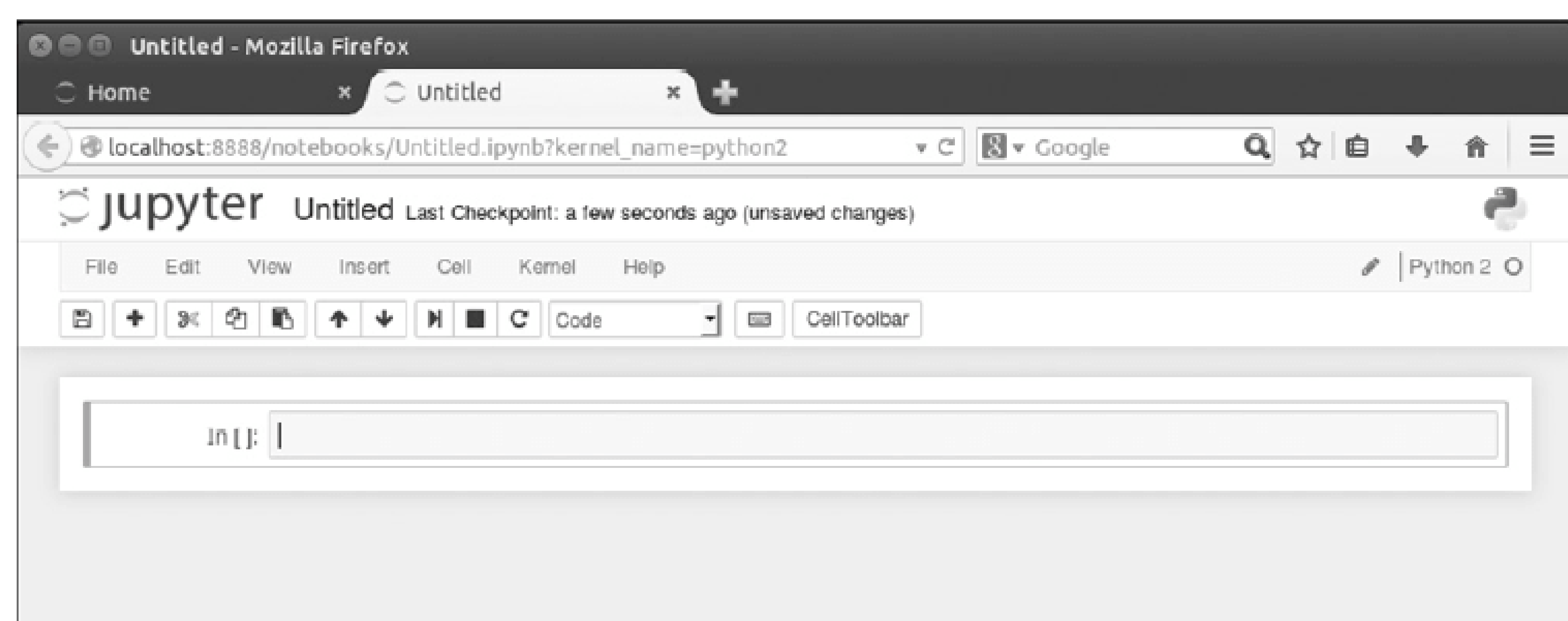


图 12-11　已新建好的 IPython Notebook

12.6 如何准备数据

接下来，ALS 训练数据格式是 Rating RDD 数据类型，如图 12-12 所示。



图 12-12　使用 sc.textFile 将文本文件 u.data 导入 rawUserData

步骤 01 配置文件读取路径

在 IPython Notebook 输入下列指令，配置文件读取路径，如图 12-13 所示。

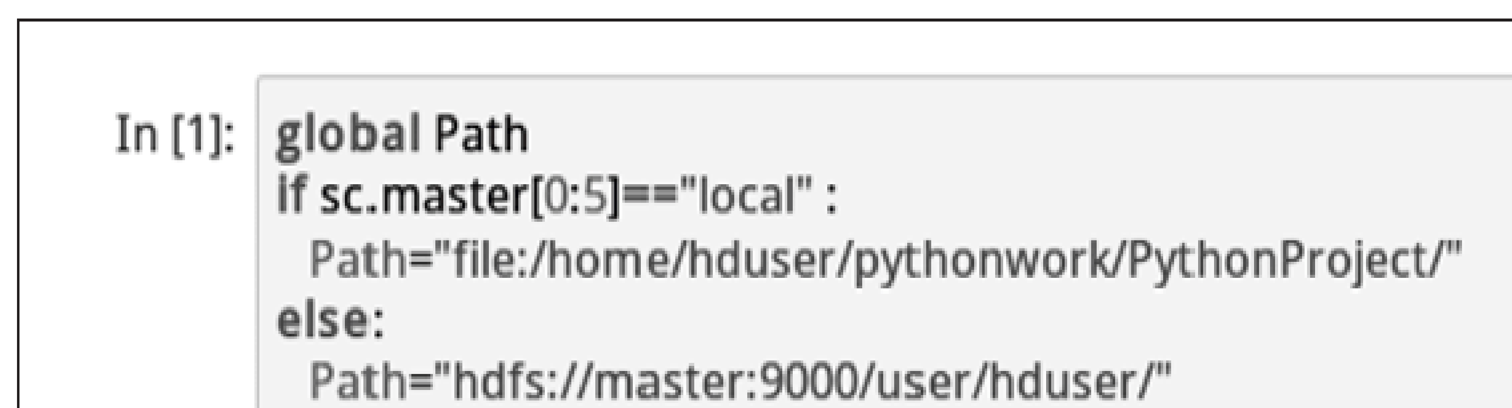


图 12-13　配置文件读取路径

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，代表当前在本地运行，读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，就有可能是 YARN Client 或 Spark Stand Alone，必须读取 HDFS 文件。

步骤 02 导入 ml-100k 数据

我们将使用 sc.textFile 读取 ml-100k 数据集的 u.data，并且查看数据项数，如图 12-14 所示。

```
In [2]: rawUserData = sc.textFile(Path+"data/u.data")
        rawUserData.count()

Out[2]: 100000
```

图 12-14　导入 ml-look 数据

从以上运行结果看共有 100000 项评分数据。

步骤 03 查看 u.data 第一项数据

查看 u.data 第一项数据（见图 12-15）。

```
In [3]: rawUserData.first()

Out[3]: u'196\t242\t3\t881250949'
```

图 12-15　查看 u.data 第一项数据

以上 4 个字段分别是：用户 id、项目 id、评价、日期时间。

步骤 04 Import Rating 模块

导入 Rating 模块，如图 12-16 所示。

```
In [30]: from pyspark.mllib.recommendation import Rating
```

图 12-16　导入 Rating 模块

步骤 05 读取 rawUserData 前 3 个字段

接下来，我们要读取 rawUserData 前 3 个字段，按照用户、产品、用户对此产品的评价来编写 rawRatings，如图 12-17 所示。

```
In [7]: rawRatings = rawUserData.map(lambda line: line.split("\t")[:3] )
        rawRatings.take(5)
Out[7]: [[u'196', u'242', u'3'],
         [u'186', u'302', u'3'],
         [u'22', u'377', u'1'],
         [u'244', u'51', u'2'],
         [u'166', u'346', u'1']]
```

图 12-17　读取 rawUserData 前 3 个字段

以上运行结果显示了前 5 项 rawRatings 数据。上列命令的详细说明如表 12-4 所示。

表 12-4　各命令的详细说明

命令	说明
rawRatings=rawUserData.map(...)	使用 map 方法对 rawUserData 每一项数据进行转换，并将结果存入 rawRatings
lambda line:	定义匿名函数，传入每一项数据 line 作为参数
line.split("\t")	每一项数据以 Tab 键分隔字段，获取每一个字段
[:3]	读取前 3 个字段
rawRatings.take(5)	读取前 5 项数据

步骤 06 准备 ALS 训练数据

ALS 训练数据格式是 Rating RDD 数据类型，Rating 定义如下。

```
Rating(user, product, rating)
```

各字段说明如表 12-5 所示。

表 12-5　字段说明

字段	说明
user	用户
product	产品
rating	用户对此产品评价

可以使用下列指令编写 ratingsRDD（见图 12-18）。

```
In [52]: ratingsRDD = rawRatings.map(lambda x: (x[0],x[1],x[2]))
         ratingsRDD .take(5)
Out[52]: [(u'196', u'242', u'3'),
          (u'186', u'302', u'3'),
          (u'22', u'377', u'1'),
          (u'244', u'51', u'2'),
          (u'166', u'346', u'1')]
```

图 12-18　编写 ratingsRDD

从以上运行结果可以看到前 5 项数据，并且每一项含有用户、产品、用户对此产品评价。上列命令的详细说明如表 12-6 所示。

表 12-6　各命令的详细说明

命令	详细说明
ratingsRDD=rawRatings.map(	rawRatings 使用map 将每一项数据进行转换，并且返回 Rating 格式数据类型
lambda x: (x[0], x[1], x[2]))	lambda 匿名函数语法, 传入x 参数: x[0] 用户 x[1] 产品 x[2] 用户对此产品的评价
ratingsRDD.take(5)	读取前5 项数据

步骤 07 查看 ratingsRDD 项数

可以使用 .count() 查询数据项数（见图 12-19）。

```
In [53]: numRatings = ratingsRDD.count()
         numRatings
Out[53]: 100000
```

图 12-19　查看 ratingsRDD 项数

以上运行结果显示共有 100000 项评价。

步骤 08 查看不重复用户数

x[0] 是用户字段，可以先使用 .map(lambda x: x[0]) 转换为用户数据，再使用 .distinct() 筛选出不重复的数据，最后显示 numUsers，如图 12-20 所示。

```
In [55]: numUsers = ratingsRDD.map(lambda x: x[0] ).distinct().count()
         numUsers
Out[55]: 943
```

图 12-20　查看不重复用户数

以上运行结果显示共有 943 个不重复用户。

步骤 09 查看不重复电影数

x[0] 是电影字段，可以先使用 .map(lambda x: x[1]) 转换为电影数据，再使用 .distinct() 筛选出不重复的数据，如图 12-21 所示。

```
In [57]: numMovies = ratingsRDD.map(lambda x: x[1]).distinct().count()
         numMovies
Out[57]: 1682
```

图 12-21　查看不重复电影数

12.7 如何训练模型

如图 12-22 所示，我们将使用 rawUserData 数据以 map 转换为 rawRatings，再改用 map 转换为 ALS 训练数据格式 RDD[Rating]。然后使用 ALS.train 进行训练，训练完成后就会创建推荐引擎模型 MatrixFactorizationModel。

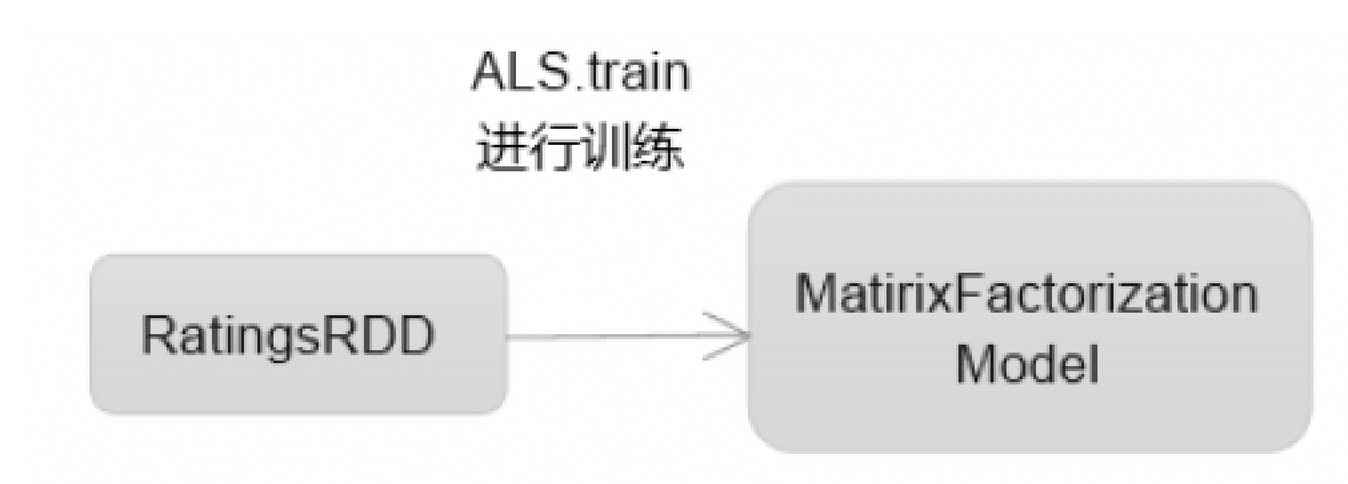


图 12-22　使用 ALS.train 进行训练的示意图

步骤 01 导入（Import）ALS 模块（见图 12-23）

```
In [16]: from pyspark.mllib.recommendation import ALS
```

图 12-23　导入 ALS 模块

步骤 02 使用 ALS.train 介绍

我们将使用 ALS.train 命令进行训练。ALS.train 可分为显式评分训练与隐式评分训练，参数说明如表 12-7 所示。

➢ 显式评分（Explicit Rating）训练

ALS.train(ratings, rank, iterations=5, lambda_=0.01):
返回 MatrixFactorizationModel

➢ 隐式评分（Implicit Rating）训练

ALS.trainImplicit (ratings, rank, iterations=5, lambda_=0.01):
返回 MatrixFactorizationModel
两种评分训练的作用都是训练数据并返回模型。

表 12-7　ALS.train 命令参数说明

参数	说明
ratings	训练的数据格式是 Rating (userID, productID, rating) 的 RDD
rank	rank 指的是当我们矩阵分解 Matrix Factorization 时，将原本矩阵 $A(m\times n)$分解成 $X(m\times \text{rank})$ 矩阵与 $Y(\text{rank}\times n)$矩阵

（续表）

参数	说明
Iterations	ALS 算法重复计算次数（默认值 5）
lambda	默认值为 0.01
返回数据	返回训练完成的模型，数据类型是 MatrixFactorizationModel

这些参数值的设置会影响结果的准确度以及训练所需的时间。后续章节会说明如何调校找出最佳的参数组合。

➢ **训练完成后会产生 MatrixFactorizationModel 模型**

训练时会执行矩阵分解（Matrix Factorization），完成后会将原本矩阵 $A(m\times n)$分解成 $X(m\times \text{rank})$ 矩阵与 $Y(\text{rank}\times n)$矩阵，详细说明如表 12-8 所示。

表 12-8　MatrixFactorizationModel 对象

Member 成员	说明
rank	分解的参数
userFeatures	分解后用户矩阵 $X(m\times \text{rank})$
productFeatures	分解后产品矩阵 $Y(\text{rank}\times n)$

步骤 03 使用 ALS.train 命令进行训练

本范例使用显式评分（Explicit rating）训练。我们可以使用 ALS.train 命令并传入之前创建的 ratingsRDD 进行训练，训练完成后返回 model，如图 12-24 所示。

```
In [18]: model = ALS.train(ratingsRDD, 10, 10, 0.01)
         print model
         <pyspark.mllib.recommendation.MatrixFactorizationModel object at 0x7f8252c6d1d0>
```

图 12-24　使用 ALS.train 命令进行训练

从以上运行结果可以看出，训练完成后建立的 model 是 MatrixFactorizationModel 数据类型。

12.8 如何使用模型进行推荐

在前面的章节中我们已经训练完成并建立模型，接下来将使用此模型进行推荐。

步骤 01 **针对用户推荐电影**

我们可以针对每一个会员定期发送短信或 E-mail，或在会员登录时向会员推荐可能会感兴

趣的电影。

针对用户推荐电影，我们可以使用 model.recommendProducts 方法来推荐，说明如表 12-9 所示。

MatrixFactorizationModel.recommendProducts(user: Int, num: Int)：输入参数 user，针对此 user 推荐给他/她有可能感兴趣的产品。

表 12-9　参数说明

参数	说明
user	要被推荐的用户 id
num	推荐的项数
返回	返回 Rating 数据类型(user: Int, product: Int, rating: Double)的数组：List 中每一项都是系统的推荐产品，rating 是系统给的评分，rating 越高代表系统越加优先向此用户推荐此产品；返回的 List 会按 rating 从大到小排序，也就是说第一项是系统推荐评分最高的产品

下面给出一个以上方法的使用范例。例如，针对用户 196 推荐前 5 部电影：

➢ **针对用户推荐电影**

使用训练完成模型进行推荐，传入参数（user=100, num=5），如图 12-25 所示。

```
In [17]: model.recommendProducts(100,5)
Out[17]: [Rating(user=100, product=1141, rating=6.388971603110022),
          Rating(user=100, product=394, rating=5.919113143619917),
          Rating(user=100, product=1192, rating=5.655175967094093),
          Rating(user=100, product=723, rating=5.452007348836167),
          Rating(user=100, product=1006, rating=5.3733955267679665)]
```

图 12-25　传入参数

以上执行结果显示，第 1 项数据是系统针对此用户首先推荐的产品。其意义是推荐给用户 ID 100，产品 ID 1141，推荐评分大约为 6.38。推荐评分越高，代表系统越优先推荐此产品。

步骤 02　查看针对用户推荐产品的评分

我们可以查询系统对用户推荐产品的评分。例如，我们查询上一步骤的第一项，系统针对用户 100 推荐产品 1141 的评分，如图 12-26 所示。

```
In [19]: model.predict(100, 1141)
Out[19]: 6.388971603110022
```

图 12-26　查看针对用户推荐产品的评分

步骤 03　针对电影推荐给用户

当我们想要促销某些电影时，可以找出可能会对这些电影感兴趣的会员，并且发送短信或 E-mail。针对电影推荐给用户，我们可以使用 model.recommendUsers 方法推荐：

MatrixFactorizationModel.recommendUsers(product: Int, num: Int)：输入参数 product，针对此 product 推荐给可能有兴趣的用户。

参数说明如表 12-10 所示。

表 12-10　参数说明

参数	说明
product	要被推荐的产品 id
num	推荐的项数
返回	返回 Rating 数据类型(user: Int, product: Int, rating: Double)的 List：List 中每一项都是系统针对产品推荐给用户，rating 是系统给的评分，rating 越高代表针对此产品越优先推荐给用户；返回的 List 会以 rating 从大到小排序，也就是说第一项是系统推荐最有兴趣的用户

可以使用下列指令：

➢ **针对电影 200 推荐前 5 个用户**

使用训练完成模型进行推荐，传入参数（product=200,num=5)，如图 12-27 所示。

```
In [20]: model.recommendUsers(product=200,num=5)
Out[20]: [Rating(user=153, product=200, rating=7.201407734348801),
          Rating(user=857, product=200, rating=6.8806830828459695),
          Rating(user=820, product=200, rating=6.789958454867682),
          Rating(user=762, product=200, rating=6.633066203093734),
          Rating(user=143, product=200, rating=6.415461132252602)]
```

图 12-27　针对电影 200 推荐前 5 个用户

以上运行结果显示，第一项数据是系统针对电影推荐给用户。其意义是针对电影 ID 200 推荐给用户 ID 153，推荐评分大约为 7.2。

12.9 显示推荐的电影名称

之前的范例只显示推荐电影的 ID，接下来将介绍如何显示推荐电影的名称。

步骤 01 读取 u.item 文本文件

接下来，我们使用 sc.textFile 将 u.item 文本文件导入 itemRDD，如图 12-28 所示。

```
In [22]: itemRDD = sc.textFile(Path+"data/u.item")
         itemRDD.count()
Out[22]: 1682
```

图 12-28　读取 u.item 文本文件

以上运行结果共计 1682 项电影数据。

步骤 02 创建“电影 ID 与名称”的字典（或对照表）

为了显示推荐电影的名称，必须创建“电影 ID 与名称”的字典（对照表），如图 12-29 所示。

```
In [23]: movieTitle= itemRDD.map( lambda line : line.split("|"))     \
                        .map(lambda a: (float(a[0]),a[1]))           \
                        .collectAsMap()
         len(movieTitle)
Out[23]: 1682
```

图 12-29 创建“电影 ID 与名称”的字典

以上运行结果显示“电影 ID 与名称”字典共计 1682 项。

上述命令说明如表 12-11 所示。

表 12-11 命令说明

命令	说明
movieTitle= itemRDD	itemRDD 经过一系列命令计算，将结果存入 movieTitle 字典
.map(lambda line : line.split("|"))	使用 map 方法针对每一项数据进行转换： 每一项数据使用 line.split("\\\|") 以 | 符号分隔字段
.map(lambda a: (float(a[0]),a[1]))	使用 map 方法将上一条命令读取的前两个字段进行转换： float(a [0]) 将第一个字段转换为 float，也就是电影 ID；a[1] 是第二个字段，为电影的名称
.collectAsMap()	使用 collectAsMap()创建 movieTitle 电影 ID / 名称字典
len(movieTitle)	显示字典项数

步骤 03 显示字典的前 5 项数据

之前我们已经创建 movieTitle 字典，现在可以使用下述命令查看对照表前 5 条数据。首先使用 movieTitle.items() 获取 movieTitle 电影 ID / 名称字典，并使用 [:5] 获取前 5 项数据，如图 12-30 所示。

```
In [24]: movieTitle.items()[:5]
Out[24]: [(1.0, u'Toy Story (1995)'),
          (2.0, u'GoldenEye (1995)'),
          (3.0, u'Four Rooms (1995)'),
          (4.0, u'Get Shorty (1995)'),
          (5.0, u'Copycat (1995)')]
```

图 12-30 显示字典的前 5 项数据

从以上运行结果可以看出第一项 MovieID 1 对应的电影名称为 Toy Story(1995)。

步骤 04 查询电影名称

有了字典之后，可以使用命令查询每一个 movie ID 的电影名称。

例如，movieTitle[5]，查询 movie ID 5 的电影名称，如图 12-31 所示。

```
In [25]: movieTitle[5]
Out[25]: u'Copycat (1995)'
```

图 12-31　查询电影名称

以上运行结果显示电影名称为 Copycat(1995)。

步骤 05　显示前 5 条推荐的电影名称

有了 movieTitle 字典，我们就可以显示推荐的电影名称，如图 12-32 所示。

```
In [30]: recommendP= model.recommendProducts(100,5)
         for p in recommendP:
           print "对用户 "+ str(p[0]) +\
                 "推荐电影"+ str(movieTitle[p[1]]) + \
                 "推荐评分"+ str(p[2])

对用户100推荐电影War Room, The (1993)推荐评分6.38897160311
对用户100推荐电影Radioland Murders (1994)推荐评分5.91911314362
对用户100推荐电影Boys of St. Vincent, The (1993)推荐评分5.65517596709
对用户100推荐电影Boys on the Side (1995)推荐评分5.45200734884
对用户100推荐电影Until the End of the World (Bis ans Ende der Welt) (1991)推荐评分5.
37339552677
```

图 12-32　显示前 5 条推荐的电影名称

以上运行结果针对用户 100 推荐 5 部电影名称。

上述命令的详细说明如表 12-12 所示。

表 12-12　命令说明

命令	说明
recommendP= model.recommendProducts(100,5)	针对用户 100 推荐 5 部电影
for p in recommendP:	使用 for 读取每一项数据
print" 对用户 "+str(p[0])+ \	显示第一个字段用户
" 推荐电影 "+ str(movieTitle[p[1]])+\	使用 movieTitle 传入第二个字段电影 ID，返回电影名称
" 推荐评分 "+ str(p[2])	显示第三个字段推荐评分

12.10 创建 Recommend 项目

在之前的章节中，我们已经使用 IPython Notebook 示范了如何导入数据、创建 Rating RDD、训练模型、推荐产品，对推荐算法流程已经有了基本概念。后续我们将介绍如何创建 Spark 应

用程序以重复使用。这里我们将建立 Recommend 推荐系统，系统中有 2 个程序，说明如下：

- RecommendTrain.py

（1）数据准备阶段

读取 u.data，经过处理后产生评分数据 ratingsRDD。

（2）训练阶段

评分数据 ratingsRDD 经过 ALS.train 训练后产生模型 Model。

（3）存储模型

存储模型 Model 在本地或 HDFS 中，作为后续推荐使用。

- Recommend.py

（1）数据准备阶段

读取 u.item 经过数据处理后，产生 movieTitle（电影 ID / 名称字典）。

（2）载入模型

载入之前存储的模型 Model。

（3）推荐阶段

使用模型 Model 进行推荐，并使用 movieTitle 转换显示推荐的电影名称。Recommend 推荐系统示意图如图 12-33 所示。

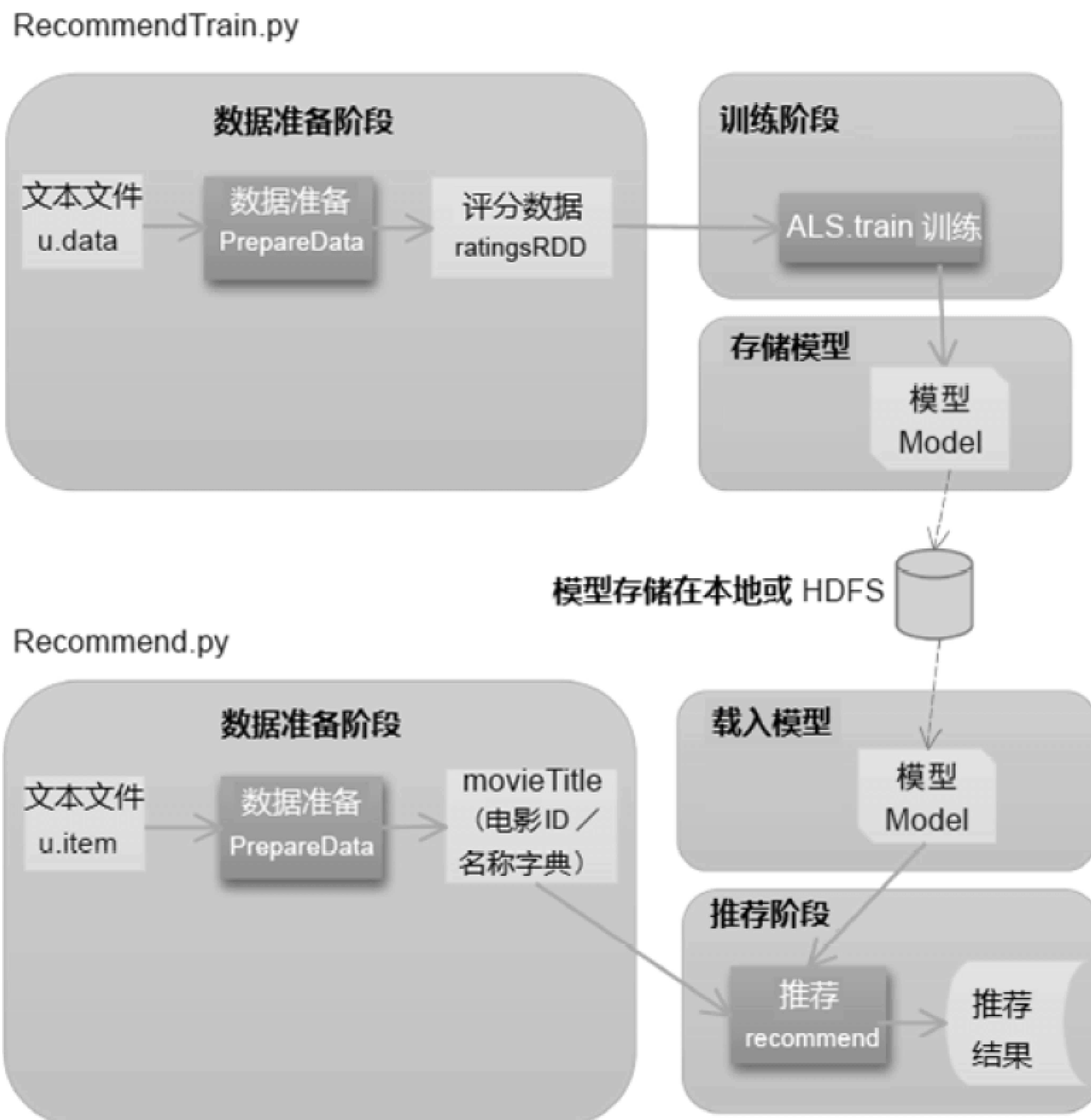


图 12-33　Recommend 推荐系统

步骤 01 启动 eclipse（见图 12-34）

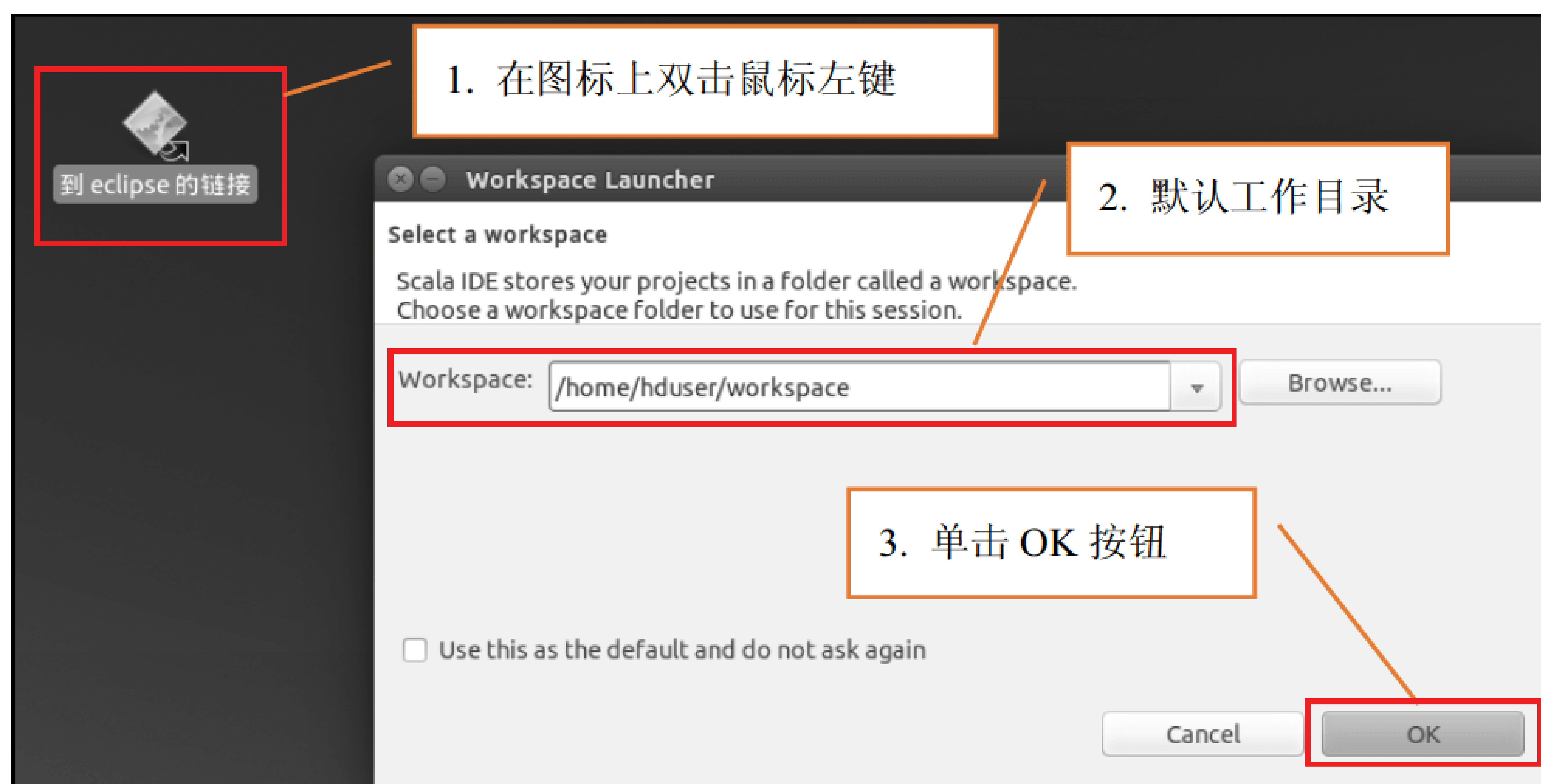


图 12-34 启动 eclipse

步骤 02 创建 Recommend 项目

参考第 11.9 节来创建新的 RecommendTrain.py 程序。

（1）用鼠标右键单击 PythonProject，出现快捷菜单，依次选择 New→File 菜单选项。

（2）打开 New File 对话框后，输入“RecommendTrain.py”，然后单击 Finish 按钮。

创建好 RecommendTrain.py 程序后，界面显示如图 12-35 所示。

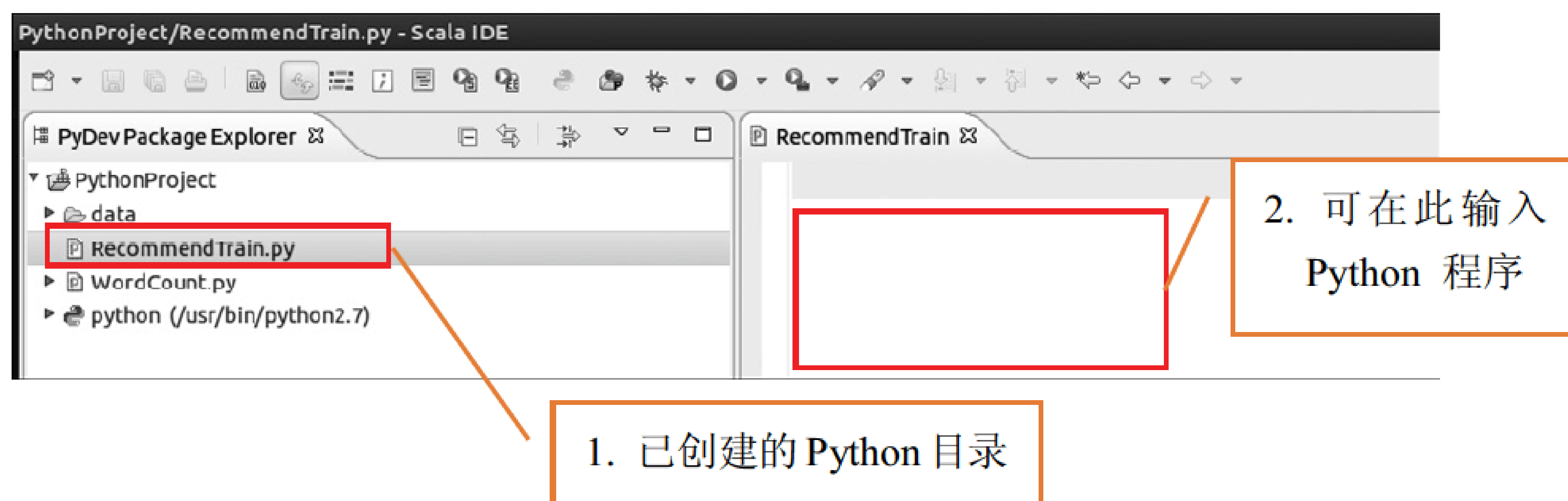


图 12-35 创建 Recommend 项目

步骤 03 main 主程序代码

编写 main 程序代码，这是程序的起点。main 程序代码分为下列 3 部分。

```
if  name  == " main ":
    sc=CreateSparkContext()
    print("========== 数据准备阶段 ===========")
    ratingsRDD = PrepareData(sc)
    print("========== 训练阶段 ===============")
```

```
    print(" 开始 ALS 训练 , 参数 rank=5,iterations=20, lambda=0.1");
    model = ALS.train(ratingsRDD, 5, 20, 0.1)
    print("========== 存储 Model========== ==")
    SaveModel(sc)
```

以上程序代码说明如表 12-13 所示。

表 12-13　main 程序代码说明

程序代码	说明
sc=CreateSparkContext()	创建 SparkConf 配置设置对象
数据准备阶段 ratingsRDD = PrepareData(sc)	读取 u.data 数据，转换为 ratingsRDD 作为后续训练数据
训练阶段 model = ALS.train(ratingsRDD, 5, 20, 0.1)	使用 ratingsRDD 执行 ALS 训练，ALS 训练，参 数 rank=5, iterations=20, lambda=0.1，返回 model 模型
存储 Model SaveModel(sc)	存储训练完成的模型，作为后续预测时使用

步骤 04 存储模型 model

之前步骤已经产生 model 模型，我们可以存储在目录中，作为后续预测时使用。输入下列程序代码，存储 model 模型。

```
def SaveModel(sc):
    try:
        model.save(sc,Path+"ALSmodel")
        print(" 已存储 Model 在 ALSmodel")
    except Exception :
        print "Model 已经存在，请先删除再存储。"
```

如果我们再次运行以上程序代码，因为模型目录已经存在，程序就会发生错误。为 了解决此问题，程序会先判断，如果存储模型的目录已经存在，会显示错误信息。在这里我们使用 try: except Exception: 语句，判断 model.save(sc,modelPath) 是否发生错误：

- 如果存储模型的目录不存在，就会运行存储模型。
- 如果存储模型的目录已经存在，就会发生错误，显示错误信息。

12.11 运行 RecommendTrain.py 推荐程序代码

在这里我们以 YARN-client 模式运行 RecommendTrain.py 进行推荐。

步骤 01 运行 RecommendTrain.py 程序代码



按照图 12-36 所示的步骤执行 RecommendTrain.py 程序代码。

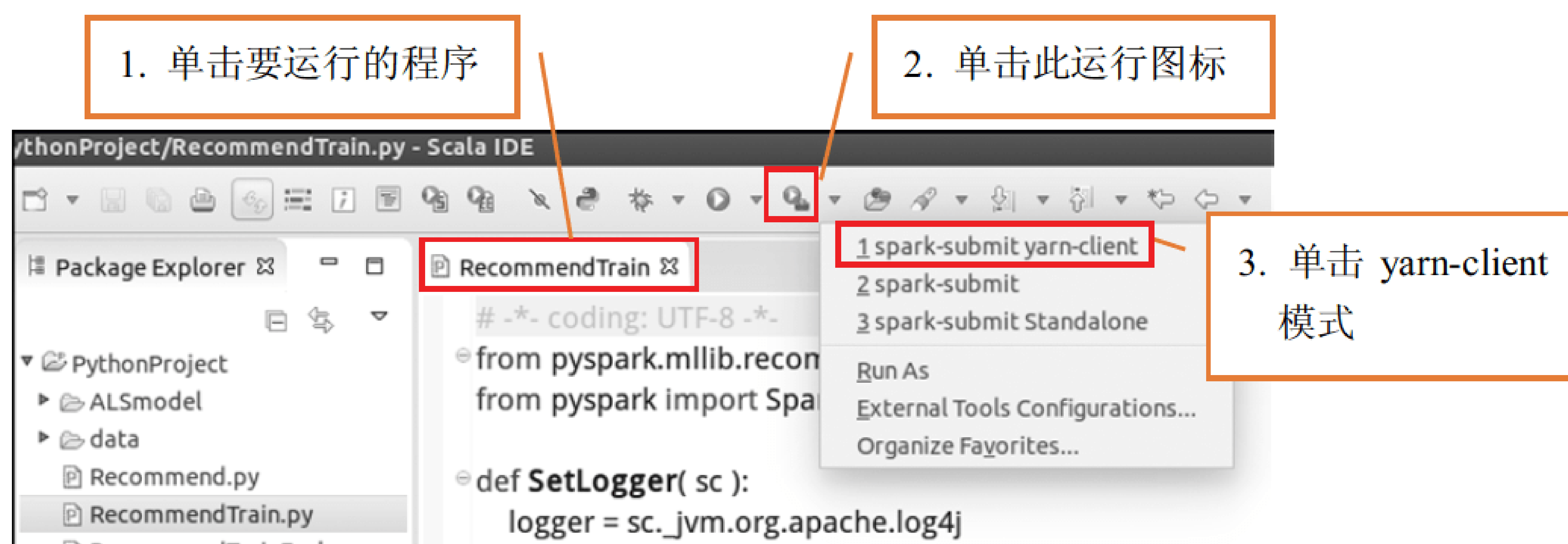


图 12-36　运行 RecommendTrain.py 程序代码

运行 Recommend.py 程序的结果如图 12-37 所示。

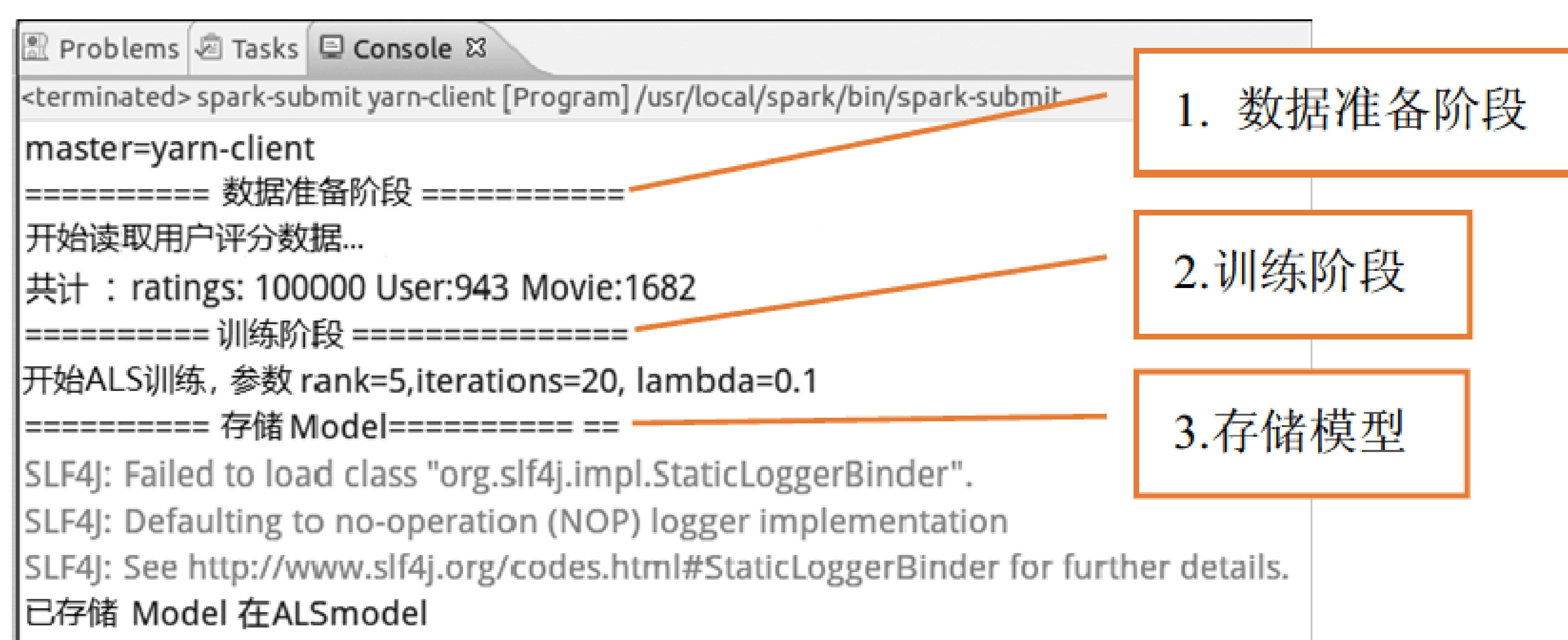


图 12-37　运行结果

步骤 02　查看存储在 HDFS 中的模型

以上程序运行后，已经将 model 模型存储在 HDFS 中。我们可以在“终端”程序输入下列命令，查看已经存储在 HDFS 中的模型。

```
hadoop fs -ls /user/hduser/ALSmodel
```

运行结果如图 12-38 所示。

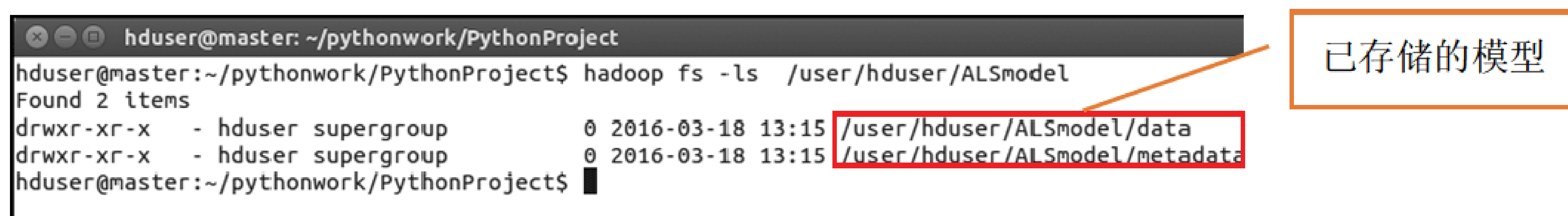


图 12-38　查看存储在 HDFS 中的模型

从图 12-38 中，我们可以看到 model 模型存储在 /user/hduser/ALSmodel，包含 data 与 metadata。

12.12 创建 Recommend.py 推荐程序代码

之前章节我们已经运行 RecommendTrain.py 存储模型，现在我们要创建 Recommend.py 读取此模型，推荐用户或电影。

步骤 01 创建 Recommend.py

请参考第 11.9 节来创建 RecommendTrain.py 程序。

（1）右击 PythonProject，出现快捷菜单，依次选择 New→File 菜单选项。
（2）打开 New File 对话框后，输入“Recommend.py”，然后单击 Finish 按钮。

创建完成的 Python 程序如图 12-39 所示。

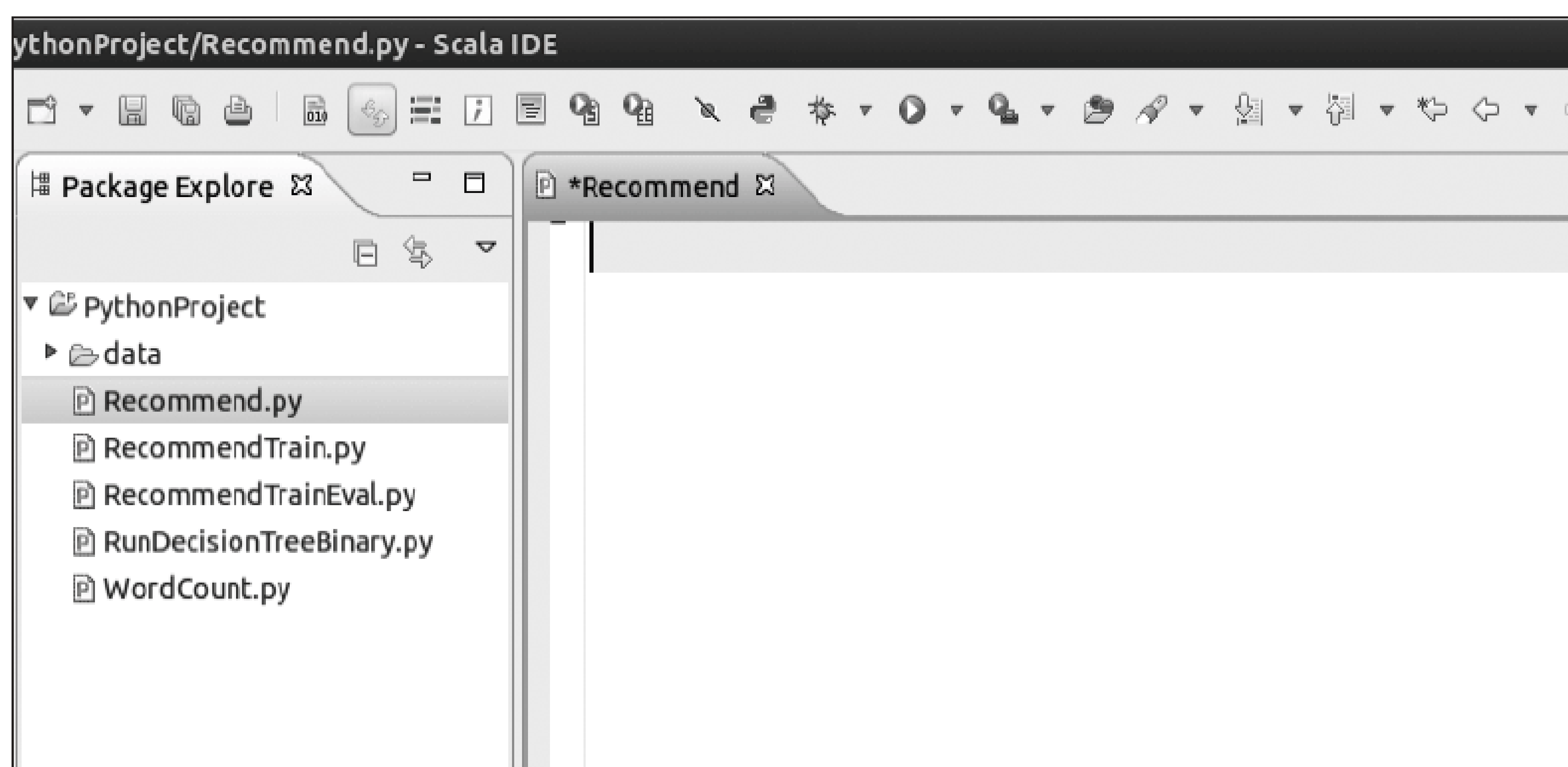


图 12-39 创建完成的 Python 程序

步骤 02 main 主程序代码

输入 main 主程序代码，如图 12-40 所示。

```
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("请输入2个参数")
        exit(-1)
    sc=CreateSparkContext()
    print("==========数据准备===============")
    (movieTitle) = PrepareData(sc)
    print("==========载入模型===============")
    model=loadModel(sc)
    print("==========进行推荐===============")
    Recommend(model)
```

图 12-40 输入主程序代码

以上程序代码说明如表 12-14 所示。

表 12-14　主程序代码说明

程序代码	说明
if len(sys.argv) != 3: print(" 请输入 2 个参数 ") exit(-1)	本程序 len(sys.argv) 必须等于 3。如果 len(sys.argv) 不等于 3，就结束执行
sc=CreateSparkContext()	创建 SparkConf 配置设置对象
数据准备阶段 (movieTitle) = PrepareData(sc)	读取 u.item 数据，转换为电影 ID 与名称字典，作为后续显示预测结果时转换电影 ID 为电影名称
载入模型 model=loadModel(sc)	载入之前存储的模型
进行推荐 Recommend(model)	使用模型进行推荐

➢ **Recommend.py 输入参数处理**

你也许会觉得奇怪，本程序只需要输入两个参数，但是为何判断 len(sys.argv) 必须等于 3？这是因为我们执行 Recommend.py 命令（例如下列指令）时，在程序中 sys.argv 会接收到 3 个值。

```
spark-submit --driver-memory 512m --executor-cores 2 --executor-memory 1g --master yarn --deploy-mode client ~/pythonwork/PythonProject/Recommend.py --U 100
```

- sys.argv[0]= '/home/hduser/pythonwork/PythonProject/Recommend.py' 程序的路径。
- sys.argv[1]='--U' 第 1 个参数，代表要执行针对用户推荐电影。
- sys.argv[2]= '100' 第 2 个参数，代表要推荐的用户 ID。

步骤 03 创建电影 ID 与名称对照表

输入图 12-41 所示的程序代码。

```
def PrepareData(sc):
    print("开始读取电影ID与名称字典 ...")
    itemRDD = sc.textFile(Path+"data/u.item")
    movieTitle= itemRDD.map( lambda line : line.split("|"))    \
                        .map(lambda a: (float(a[0]),a[1]))     \
                        .collectAsMap()
    return(movieTitle)
```

图 12-41　创建“创建电影 ID 与名称”字典

以上程序代码主要是读取 u.item 创建“电影 ID 与名称”字典（或对照表），后续我们可以用此字典将电影 ID 转换为电影名称。

步骤 04 编写 Recommend(model) 函数

执行 Recommend.py 必须输入参数，告诉程序当前我们现在要执行的功能：

- 针对此用户推荐电影。例如，执行程序 Recommend.py --U100，代表针对用户 ID=100 推荐电影。
- 针对电影推荐给用户。例如，执行程序 Recommend.py --M200，代表针对电影 ID=200 推荐给用户。

所以我们输入 Recommend(model) 函数程序代码来处理输入参数，如图 12-42 所示。

```
def Recommend(model):
    if sys.argv[1]=="--U":
        RecommendMovies(model, movieTitle,int(sys.argv[2]))
    if sys.argv[1]=="--M":
        RecommendUsers(model, movieTitle,int(sys.argv[2]))
```

图 12-42　编写 Recommend(model)函数

程序代码判断第 1 个参数 argString[1]：

- argString[1] 如果是 "--U" 就执行：

```
RecommendMovies(model, movieTitle,int(sys.argv[2]))
```

其中，int(sys.argv[2]) 读取第 2 个参数，转换为 int，代表用户 ID。
针对此用户 ID 推荐电影。

- argString[1] 如果是 "--M " 就执行：

```
RecommendUsers(model, movieTitle,int(sys.argv[2]))
```

其中，int(sys.argv[2]) 读取第 2 个参数，转换为 int 代表电影 ID。
针对此电影 ID 推荐给用户。

步骤 05 针对此用户推荐电影

创建 RecommendMovies 函数，如图 12-43 所示。针对此用户推荐电影。

```
def RecommendMovies(model, movieTitle, inputUserID):
    RecommendMovie = model.recommendProducts(inputUserID, 10)
    print("针对用户 id" + str(inputUserID) + "推荐下列电影:")
    for rmd in RecommendMovie:
        print "针对用户id {0} 推荐电影 {1} 推荐评分 {2}".\
            format( rmd[0],movieTitle[rmd[1]],rmd[2])
```

图 12-43　针对用户推荐电影

以上程序代码：

（1）定义 RecommendMovies 函数，输入参数：model 载入的模型，movieTitle 字典，inputUserID 输入的 UserID。

（2）使用 model.recommendProducts(inputUserID, 10)，针对此用户 ID 推荐 10 部电影。推 荐结果存储在 RecommendMovie 中。

（3）使用 for 读取 RecommendMovie 每一项数据，并显示推荐评分。

步骤 06　编写 RecommendUsers 函数（见图 12-44）

```
def RecommendUsers(model, movieTitle, inputMovieID) :
  RecommendUser = model.recommendUsers(inputMovieID, 10)
  print "针对电影 id {0} 电影名：{1} 推荐下列用户 id: ".\
      format( inputMovieID,movieTitle[inputMovieID])
  for rmd in RecommendUser:
    print  "针对用户 id {0}  推荐评分 {1} ".format( rmd[0],rmd[2])
```

图 12-44　写 RecommendUsers 函数

以上程序代码主要是使用 model.recommendUsers(inputMovieID, 10)针对此电影 ID 推荐给 10 个用户，推荐结果存储在 RecommendUser。然后使用 for 读取 RecommendUser 每一项数据，并显示推荐评分。

步骤 07　编写 loadModel(sc) 载入模型

输入图 12-45 所示的程序代码：编写 loadModel(sc) 载入之前的 train 模型。

```
def loadModel(sc):
  try:
    model = MatrixFactorizationModel.load(sc, Path+"ALSmodel")
    print "载入ALSModel模型"
  except Exception:
    print "找不到ALSModel模型，请先训练"
  return model
```

图 12-45　编写 loadModel(sc) 载入模型

（1）使用 try: except Exception: 判断模型文件是否存在，如果不存在，就显示错误信息。

（2）使用 MatrixFactorizationModel.load 载入 ALSModel 模型，传入参数 sc 及载入模型文件的路径。

12.13　在 eclipse 运行 Recommend.py

步骤 01　运行外部工具

按照图 12-46 所示的步骤运行 spark-submit YARN-client 外部工具。

步骤 02　在 eclipse 运行 Recommend.py YARN client 模式

在这里我们输入“--U 200”，针对用户推荐电影（见图 12-47）。

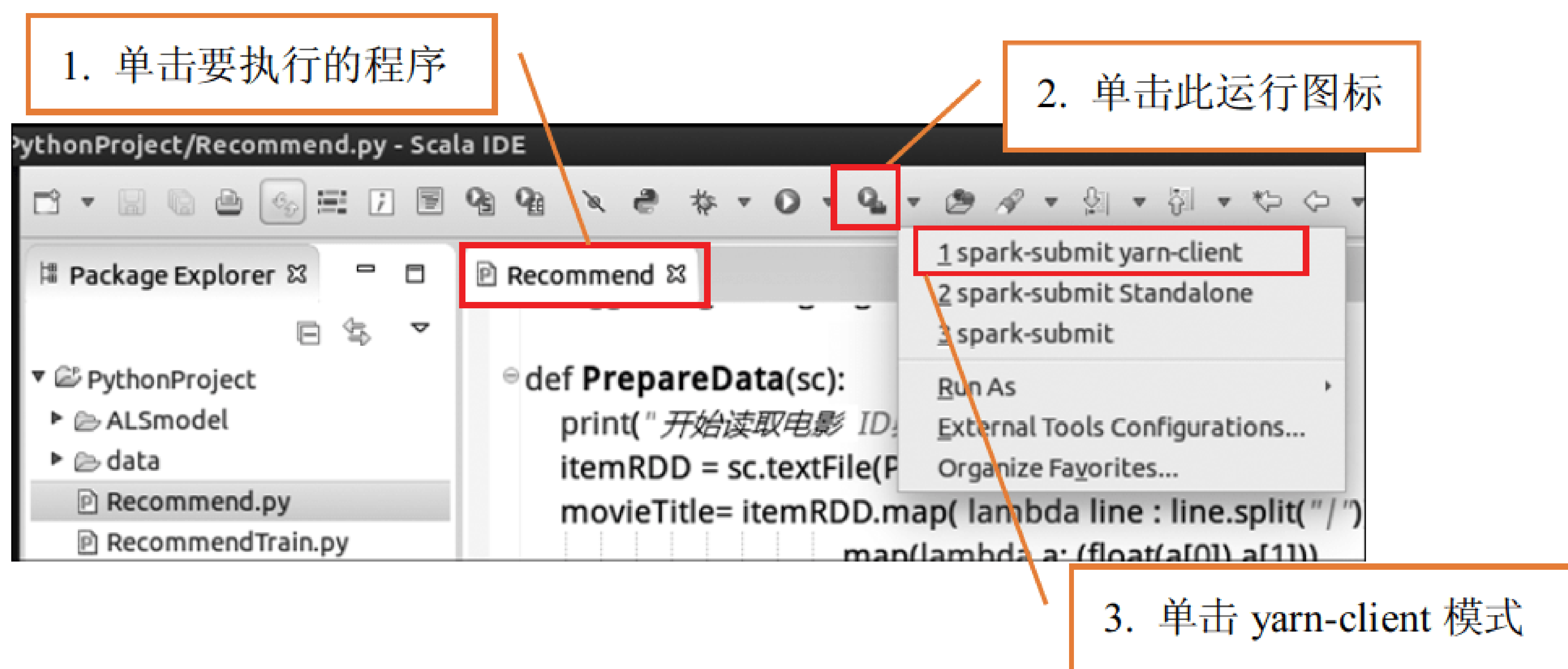


图 12-46　运行外部工具



图 12-47　针对用户推荐电影

步骤 03 针对用户推荐电影的结果

上面针对用户推荐了电影，然后输入用户 id 200，程序就会显示推荐的电影。评分越高，越会针对此用户优先推荐此电影。评分从大到小排序，也就是说第 1 项数据是系统推荐评分最高的电影，如图 12-48 所示。

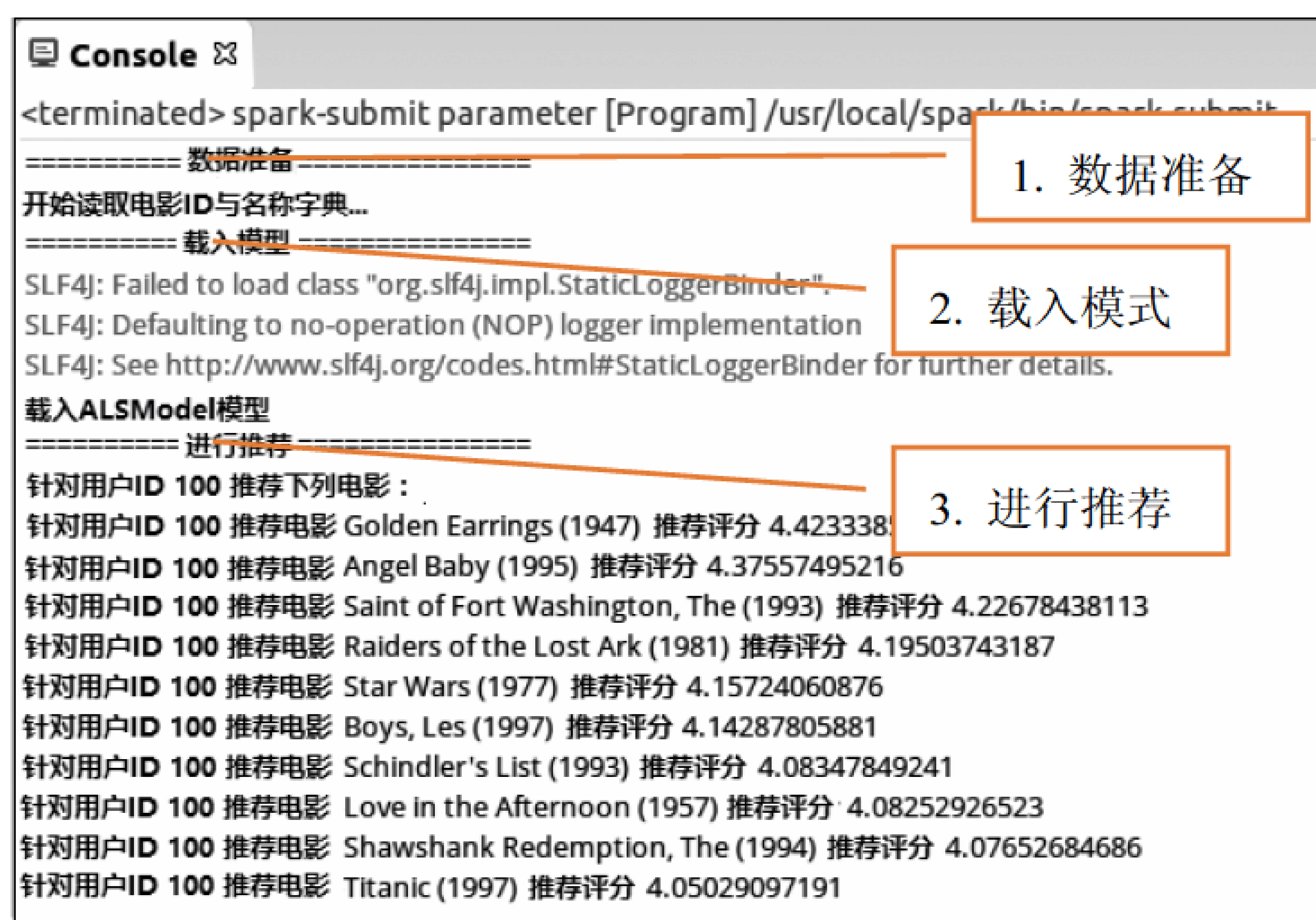


图 12-48　针对用户推荐电影的结果



步骤 04　再次运行外部工具

运行 spark-submit YARN-client 外部工具，如图 12-49 所示。

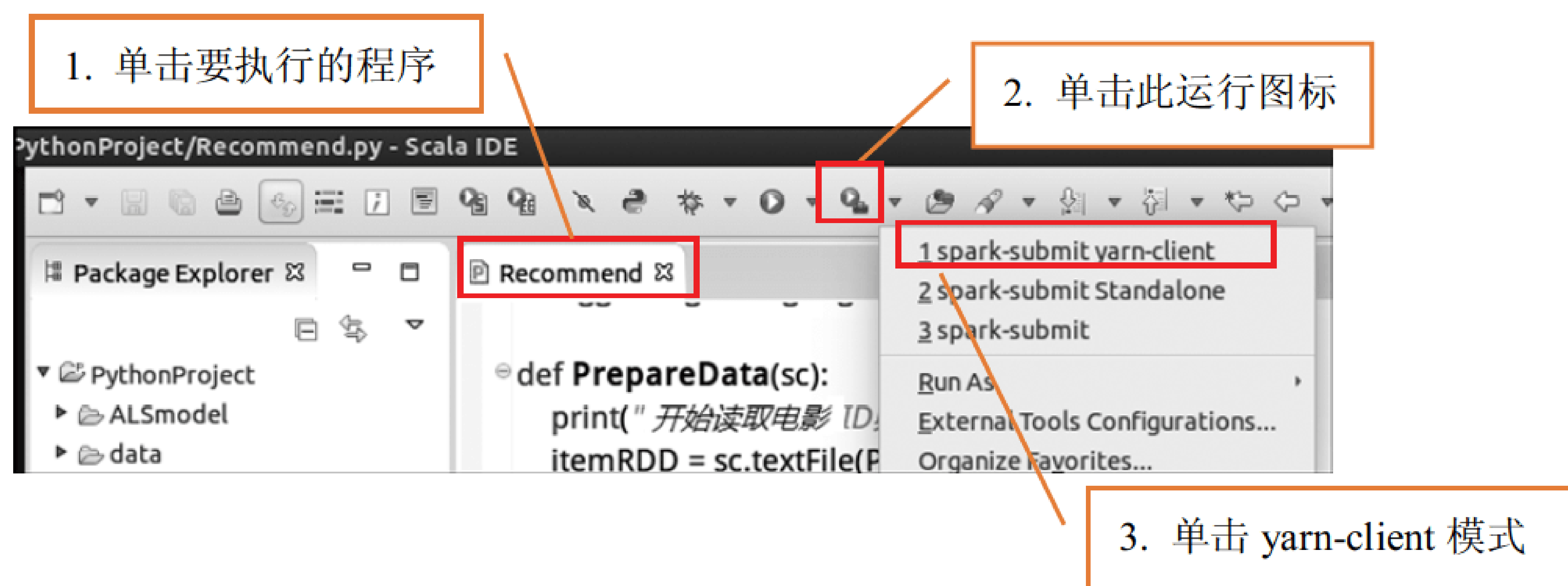


图 12-49　再次运行外部工具

步骤 05　针对电影推荐给感兴趣的用户

在这里我们输入“--M 200”，针对电影推荐有兴趣的用户，如图 12-50 所示。



图 12-50　针对电影推荐给感兴趣的用户

步骤 06　针对电影推荐用户的结果

针对电影推荐给感兴趣的用户后输入电影的 id：200，程序就会显示感兴趣的用户；评分越高，代表此用户越感兴趣。评分从大到小排序，也就是说第 1 项数据即为最感兴趣的用户，如图 12-51 所示。

```
Console ⊠
<terminated> spark-submit parameter [Program] /usr/local/spark/bin/sp
master=local[4]
========== 数据准备 ===============
开始读取电影ID与名称字典...
========== 载入模型 ===============
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBind
SLF4J: Defaulting to no-operation (NOP) logger implemen
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
载入ALSModel模型
========== 进行推荐 ===============
针对电影 ID 200 电影名：Shinning, The(1980) 推荐下列用户
针对用户ID 519 推荐评分 5.01478129151
针对用户ID 928 推荐评分 5.01200658357
针对用户ID 252 推荐评分 4.81628848983
针对用户ID 810 推荐评分 4.78245984174
针对用户ID 628 推荐评分 4.77465527222
针对用户ID 173 推荐评分 4.7535567821
针对用户ID 808 推荐评分 4.74415735237
针对用户ID 909 推荐评分 4.73592035058
针对用户ID 688 推荐评分 4.71956038379
针对用户ID 34 推荐评分 4.69775847673
```

1. 数据准备
2. 载入模式
3. 进行推荐

图 12-51　针对电影推荐用户的结果

12.14 结论

本章介绍了 Spark MLlib 支持的 ALS（Alternating Least Squares）推荐算法的基本原理，并且完成数据准备、训练模型、存储模型、进行推荐。下一章我们将介绍决策树二元分类。

第 13 章

Python Spark MLlib 决策树二元分类

决策树模型的应用广泛，除了分析数据、预测模型，在机器学习上也有应用。决策树的优点是条理清晰、方法简单、易于理解、适用范围广等。

本章将使用决策树二元分类分析 StumbleUpon 数据集，预测网页是暂时性的（ephemeral）或是长青的（evergreen），并且调校参数找出最佳参数组合，提高预测准确度。

13.1 决策树介绍

当我们使用决策树分类算法来训练数据后，会以 feature（特征字段）与 label（标签字段）建立决策树。例如，图 13-1 使用湿度与气压（feature 特征字段）来判断天气为“晴”或“雨”（label 标签字段，也就是我们预测的目标）。

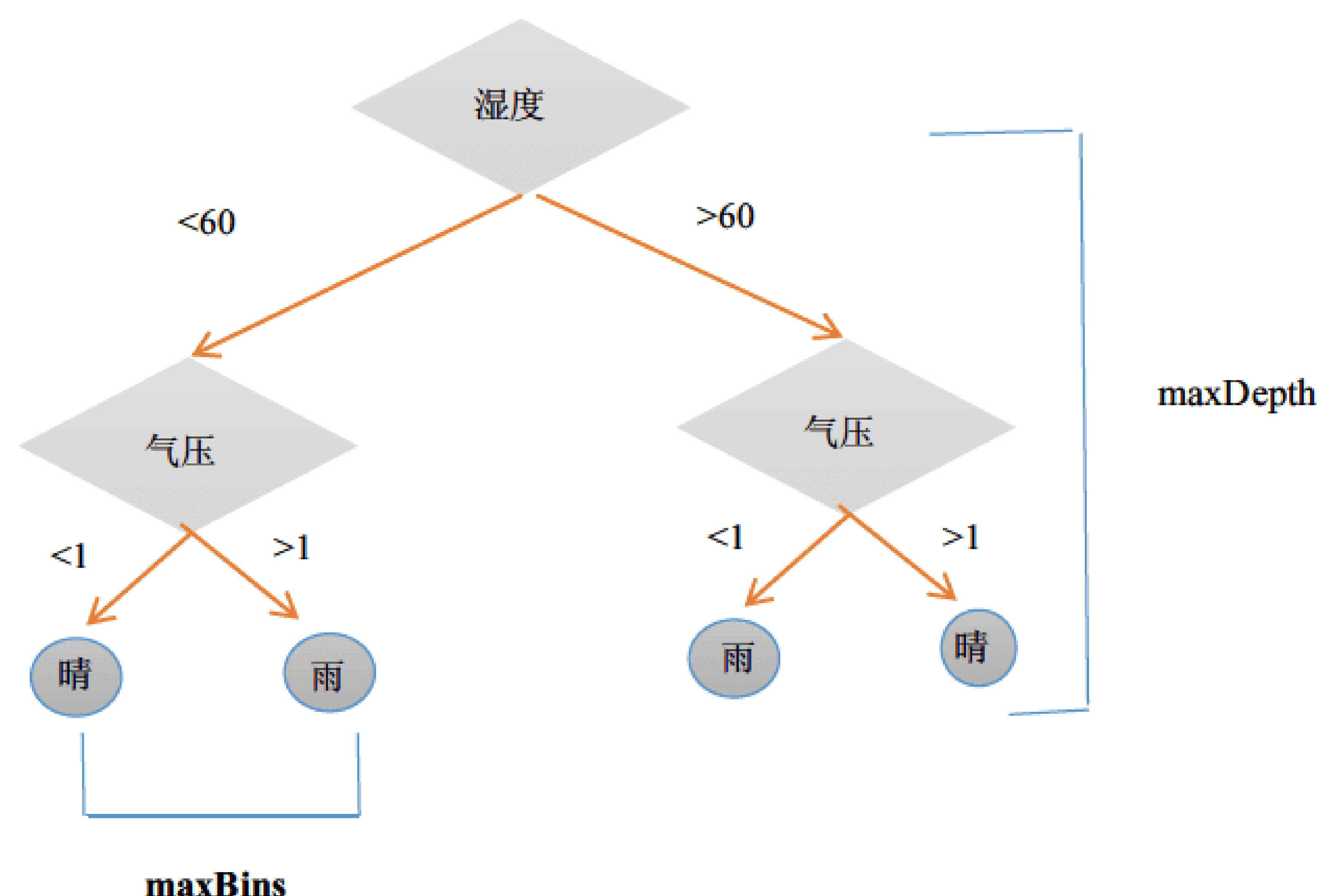


图 13-1　使用湿度与气压来判断天气为“晴”或“雨”的决策树

如图 13-1 所示，当我们使用历史数据执行训练时会建立决策树。可是决策树不可能无限成长，因此我们必须限制它的最大分支与深度，所以必须设置下列参数：

- maxBins 参数：决策树每一个节点最大分支数。
- maxDepth 参数：决策树最大深度。
- Impurity 参数：决策树分裂节点时的方法 。

当我们在父节点要分裂节点时，以什么方法作为依据呢？例如，湿度以 60 为分隔点，分为大于 60 或小于 60；或是湿度以 50 为分隔点，分为大于 50 或小于 50。到底哪一种方式比较好呢？此时有 Gini 与 Entropy 两种判断方式：

- 基尼指数（Gini）：由意大利统计学家 Corrado Gini 发明，用于计算数值散布程度（Statistical dispersion，统计离差）的指标。决策树算法对每种特征字段分隔点计算估值，选择分裂后最小的基尼指数（Gini）方式。
- 熵（Entropy）：熵（Entropy）也被用于计算系统混乱的程度。决策树算法对每种特征字段分隔点计算估值，选择分裂后最小的熵（Entropy）方式。

13.2 “StumbleUpon Evergreen”大数据问题

13.2.1　Kaggle 网站介绍

Kaggle 是一个数据分析的竞赛平台，网址为 https://www.kaggle.com/，也是 Crowdsourcing（众包）的平台。企业或研究者将大数据的问题发布到网站上，包括数据、问题说明、期望的目标、奖金等，以向大众征求解决方案。

网络上任何人都可以参与大数据问题的竞赛：下载数据、分析数据、运用机器学习、数据挖掘等知识，建立算法模型并解决问题，最后将结果上传到网站上。如果提交申请的结果符合要求，并且在参赛者中排名第一，就可以获得奖金。目前很多公司也会通过这个平台寻找大数据人才。

13.2.2　“StumbleUpon Evergreen”大数据问题场景分析

Kaggle 网站上有一个 StumbleUpon Evergreen Classification Challenge 的题目。

StumbleUpon（http://www.stumbleupon.com/）是个性化的搜索引擎，会按用户的兴趣和网页评分等记录推荐给你感兴趣的网页，例如新文章、季节菜单、新闻、教学等。超过数千万人使用 StumbleUpon 查找新网页、图片、影片……

有些网页内容是暂时性的（ephemeral），例如季节菜单、当日股市涨跌新闻等。这些文章可能只是在某一段时间会对读者有意义，过了这段时间对读者就没有意义了。有些网页内容是长青的（evergreen），例如理财观念、育儿知识等。读者会长久对这些文章感兴趣。

分辨网页是暂时性（ephemeral）或是长青的（evergreen），对于 StumbleUpon 推荐网页给用户会有很大帮助。例如，读者 A 买卖股票，他可能会对当日股市涨跌新闻感兴趣，可是过了一周就对这则新闻没兴趣了；如果是理财观念的文章，读者 A 可能会长久有兴趣。因此公司找来了大数据分析师，负责“网页分类”大数据项目。

➢ 找出问题

“问对问题”是解决问题的第一步。这些网页内容我们人类看过了，就可以大致分类为暂时性的（ephemeral）或是长青的（evergreen）。可是网页内容成千上万，我们不可能有足够的人力去判断网页是暂时性的（ephemeral）或长青的（evergreen）。不只是因为成本太高，根本就无法做到实时性。

➢ 设计解决方案模型

此时机器学习（Machine Learning）就派得上用场了。我们的目标是利用机器学习（Machine Learning），通过大量网页数据进行训练来建立一个模型，并使用这个模型去预测网页是属于暂时性的（ephemeral）或长青（evergreen）的内容。这属于二元分类问题，接下来的章节将使用二元分类算法分析 StumbleUpon 数据集。预测哪些网页是暂时性的或可以长久存在的，

并且找出最佳参数组合，提高预测准确度。

后续章节我们将以不同的机器学习分类法分析“StumbleUpon Evergreen”大数据问题，章节安排如表 13-1 所示。

表 13-1 机器学习分类法的章节安排

章节	内容
13	决策树二元分类
14	逻辑回归二元分类
15	支持向量机 SVM 二元分类
16	朴素贝叶斯二元分类

13.3 决策树二元分类机器学习

设计决策树二元分类机器学习模型面临的问题

本章我们将使用决策树二元分类设计机器学习模型，我们将面临下列问题：

（1）如何搜集数据？

StumbleUpon 过去已经累积了大量的网页数据，后续我们将介绍 StumbleUpon 数据集内容。

（2）如何进行数据准备？

StumbleUpon 数据集原始的数据是文本文件，我们必须经过一连串的处理，提取特征字段与标签字段，创建训练所需的数据格式 LabeledPoint。

（3）如何训练模型？

我们将执行 DecisionTree 训练，并且建立模型。

（4）如何使用模型进行预测？

建立 DecisionTree 模型之后，我们可以使用这个模型进行预测。

（5）如何评估模型的准确率？

有了模型之后，我们希望知道这个模型预测的准确率，必须要有一个标准来评估模型的准确率。在二元分类中我们使用 AUC 作为评估标准。

（6）模型的训练参数如何影响准确率？

在训练模型时我们会输入不同的参数，其中 DecisionTree 参数 impurity、maxDepth、maxBins 的值会影响准确率以及训练所需的时间。我们将以图表显示这些参数值，显示准确率与训练所需的时间。

（7）如何找出准确率最高的参数组合？

DecisionTree 参数 impurity、maxDepth、maxBins 有不同的排列组合，我们 将所有参数训练评估找出最好的参数组合。

（8）如何确认是否 Overfitting（过度训练）？

Overfitting（过度训练）是指机器学习所学到的模型过度贴近 trainData，从而导致误差变得很大。我们会使用另外一组数据 testData 再次测试，以避免 overfitting 的问题。如果训练评估阶段时 AUC 很高，但是测试阶段 AUC 很低，代表可能有 overfitting 的问题。如果测试与训练评估阶段的结果中 AUC 差异不大，就代表无 overfitting 的问题。

以上问题本章后续将会进行解答。

13.4 如何搜集数据

13.4.1 StumbleUpon 数据内容

可以在下列网页查看 StumbleUpon 数据的详细介绍：

```
https://www.kaggle.com/c/stumbleupon/data
```

在这里字段是以 0 为第 1 个字段。

➢ **字段 0~2**

网址、网址 ID、样板文字（见图 13-2），这些字段与判断网页是暂时性的（ephemeral）或长青的（evergreen）关系不大，所以我们会忽略。

FieldName	Type	Description
url	string	Url of the webpage to be classified
urlid	integer	StumbleUpon's unique identifier for each url
boilerplate	json	Boilerplate text

图 13-2　字段 0~2

➢ **字段 3~25**

Feature 特征字段：数值字段，内容是有关此网页的相关信息，例如网页分类、链接的数目、图像的比例等，如图 13-3 所示（字段太多只列出部分，其他请参考网页）。

alchemy_category	string	Alchemy category (per the publicly available Alchemy API found at www.alchemyapi.com)
alchemy_category_score	double	Alchemy category score (per the publicly available Alchemy API found at www.alchemyapi.com)
avglinksize	double	Average number of words in each link
commonLinkRatio_1	double	# of links sharing at least 1 word with 1 other links / # of links
commonLinkRatio_2	double	# of links sharing at least 1 word with 2 other links / # of links
commonLinkRatio_3	double	# of links sharing at least 1 word with 3 other links / # of links
commonLinkRatio_4	double	# of links sharing at least 1 word with 4 other links / # of links
compression_ratio	double	Compression achieved on this page via gzip (measure of redundancy)
embed_ratio	double	Count of number of <embed> usage
frameBased	integer (0 or 1)	A page is frame-based (1) if it has no body markup but have a frameset markup
frameTagRatio	double	Ratio of iframe markups over total number of markups
hasDomainLink	integer (0 or 1)	True (1) if it contains an <a> with an url with domain
html_ratio	double	Ratio of tags vs text in the page

图 13-3　字段 3~25

➢ 字段 26

这是 label，具有两个值（见图 13-4）。

- 1：代表长青（evergreen）——此网页会持续让用户感兴趣。
- 0：代表 non-evergreen——此网页具有暂时性。

label	integer (0 or 1)	User-determined label. Either evergreen (1) or non-evergreen (0); available for train.tsv only

图 13-4　字段 26

13.4.2　下载 StumbleUpon 数据

步骤 01 到下载网址

在浏览器输入下列网址，进入 Kaggle 网站的 StumbleUpon 页面：

```
https://www.kaggle.com/c/stumbleupon/data
```

在网页中，可以看到两个文件：train.tsv（训练数据）、test.tsv（测试数据）。我们将下载这两个文件。先示范如何下载 train.tsv（训练数据），如图 13-5 所示。

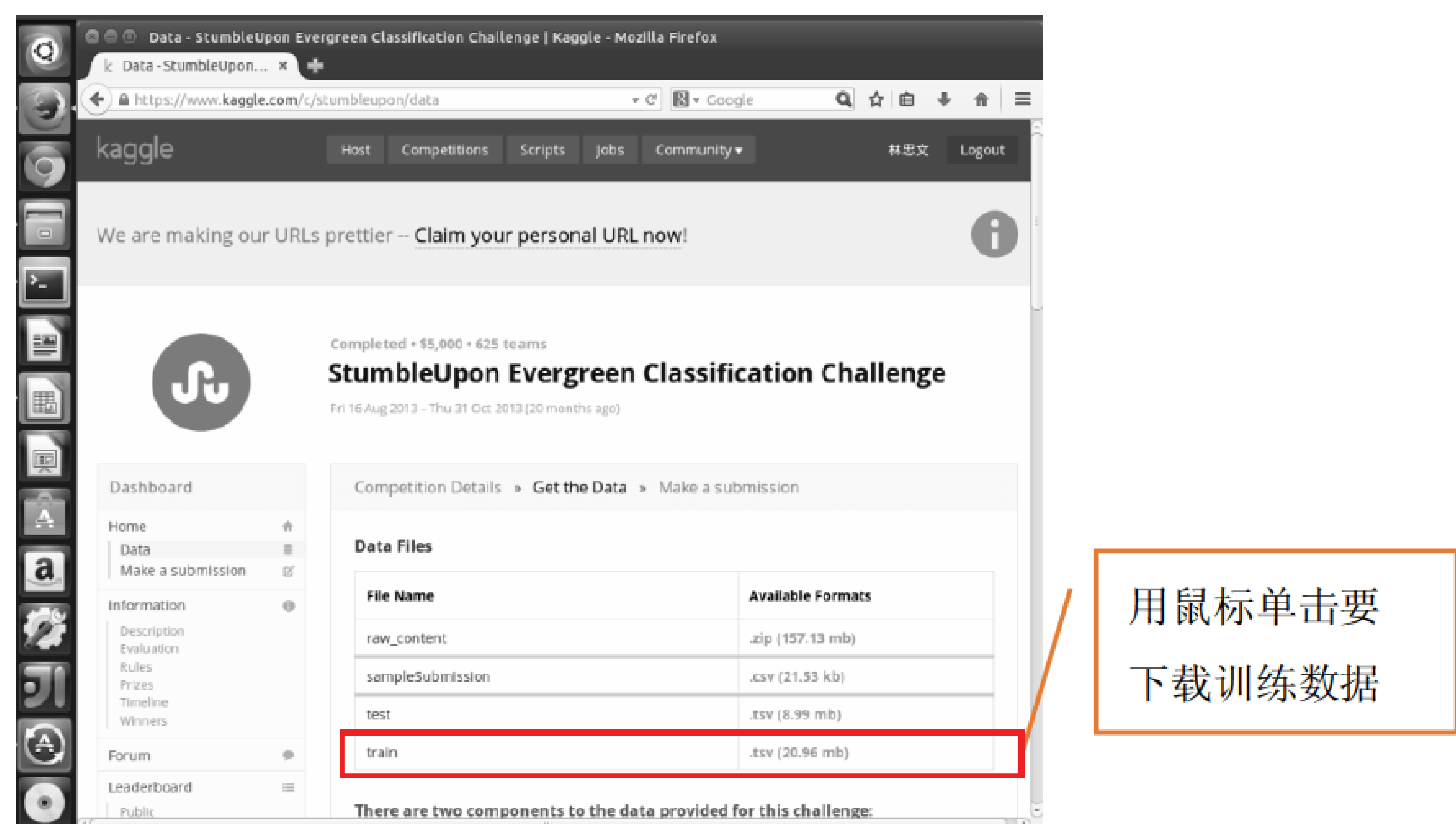


图 13-5　选择下载 train.tsv 文件

步骤 02　注册网站

下载前必须先注册，可以使用邮箱账号进行注册，如图 13-6 所示。

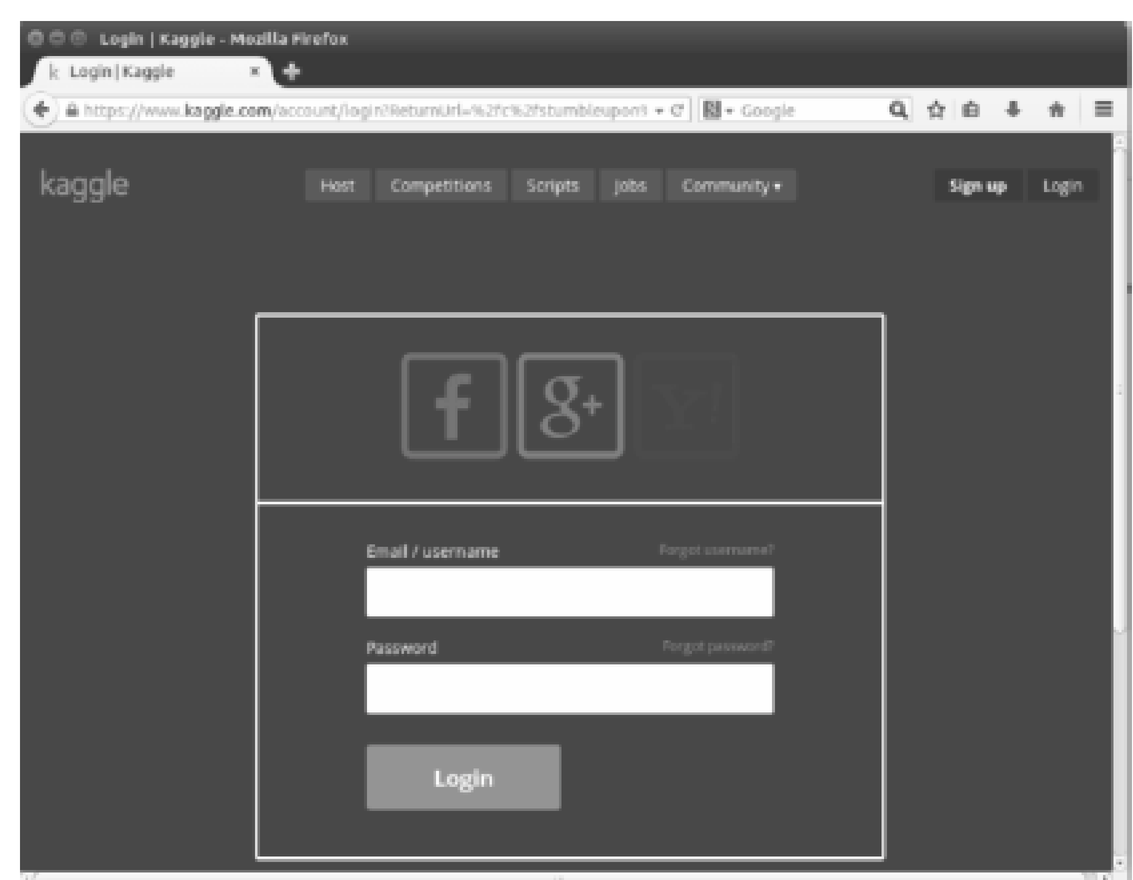


图 13-6　注册

步骤 03　下载 train.tsv（见图 13-7）

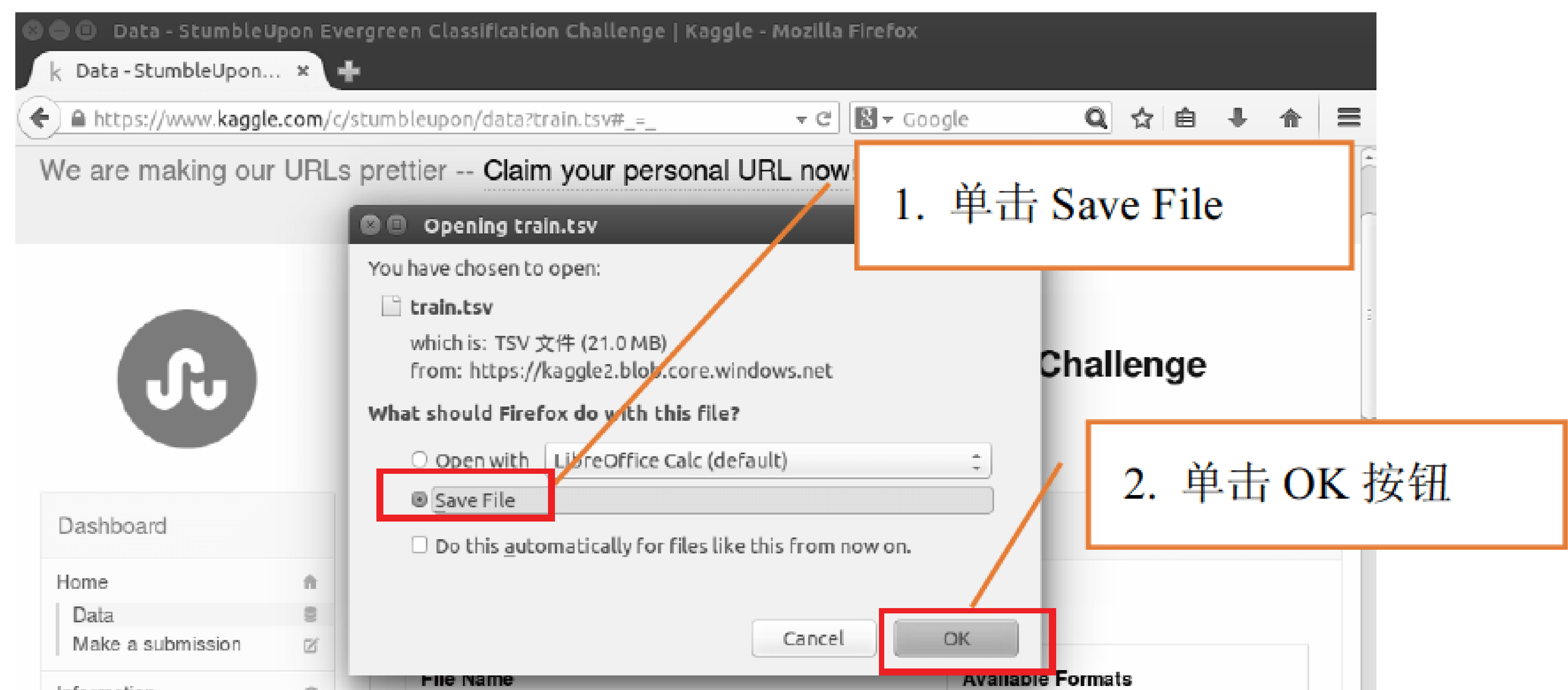


图 13-7　下载 train.tsv

步骤04 打开所在的文件夹（见图 13-8）

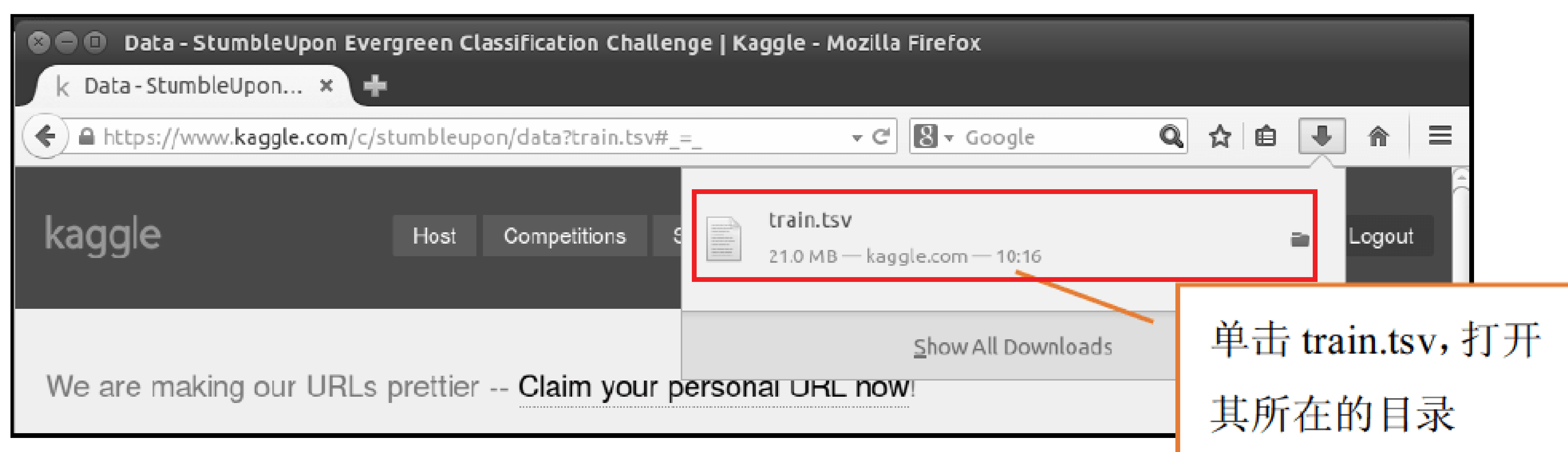


图 13-8 打开所在的文件夹

步骤05 下载 test.tsv

按照类似步骤下载 test.tsv 测试数据，如图 13-9 所示。

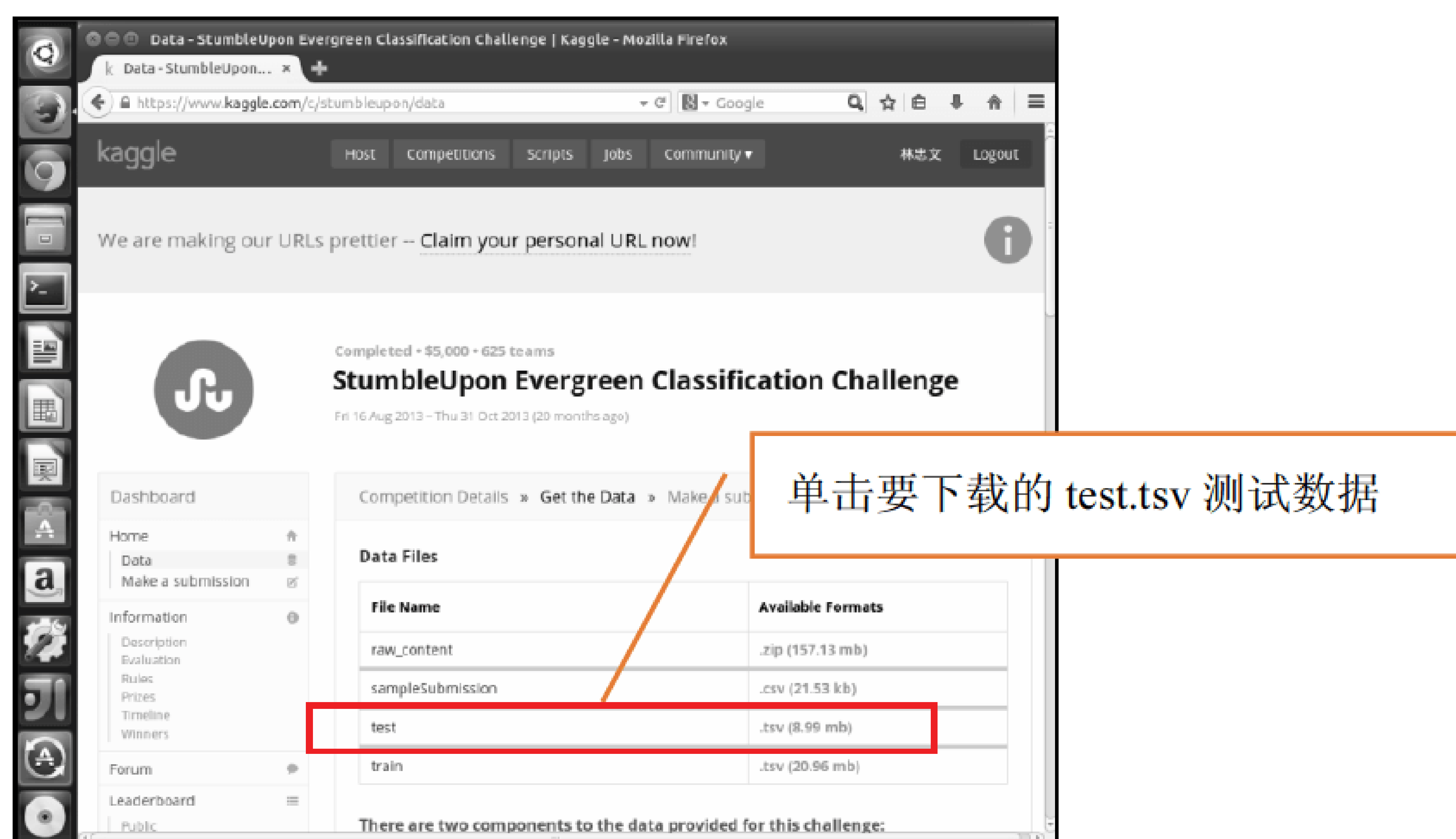


图 13-9 选择 test.tsv 文件

步骤06 查看已下载的文件

全部下载完成后可以看到两个文件：test.tsv（测试数据）、train.tsv（训练数据），如图 13-10 所示。

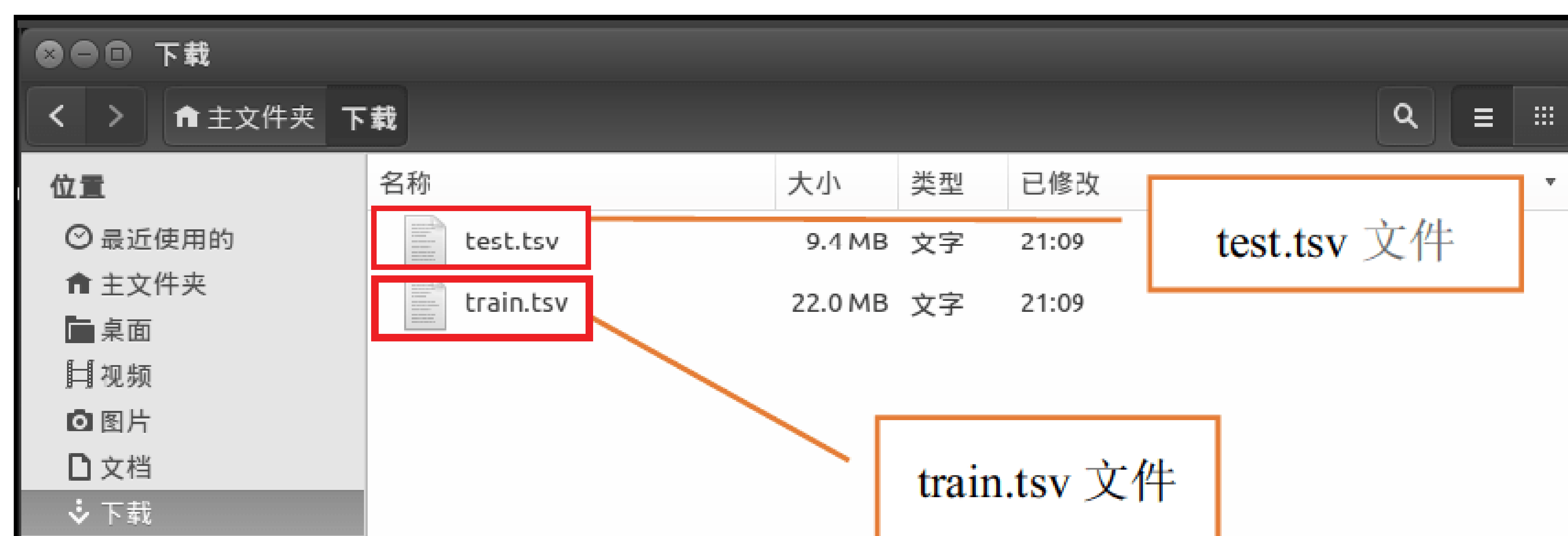


图 13-10 查看已下载的文件

13.4.3 用 LibreOffice Calc 电子表格查看 train.tsv

步骤 01 打开 train.tsv 文件（见图 13-11）



图 13-11 打开 train.tsv 文件

步骤 02 导入 train.tsv 文字到 LibreOffice Calc 电子表格

tsv 文件默认会以 LibreOffice Calc 电子表格打开。tsv 文件的字段之间是以制表符分隔的，在“导入文字”界面进行参数设置，如图 13-12 所示。



图 13-12 设置参数

步骤 03 查看已经打开的 train.tsv

打开后的屏幕显示界面如图 13-13 所示。

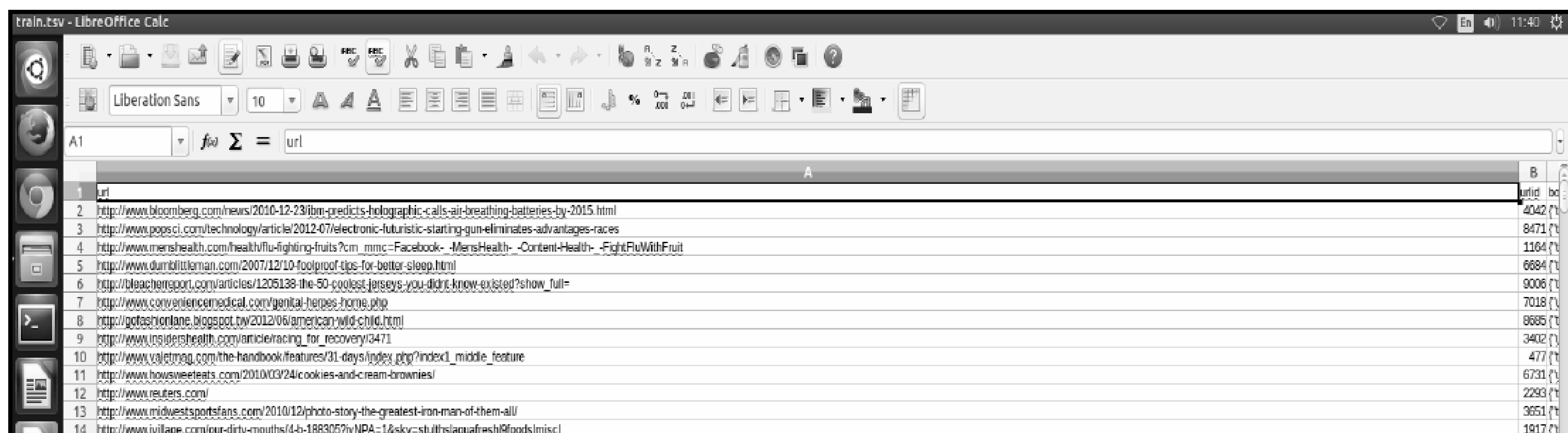


图 13-13　查看已经打开的 train.tsv

步骤 04　设置显示宽度

因为 url 字段太宽会影响阅读，所以设置所有字段固定宽度，如图 13-14 所示。

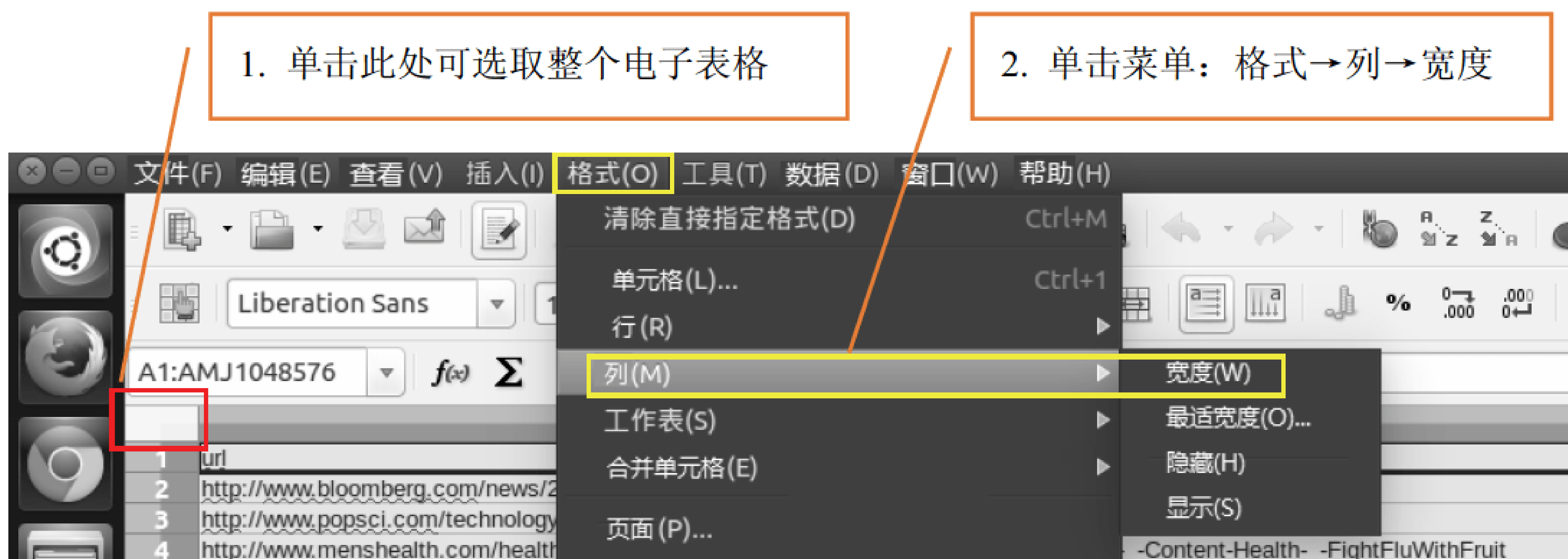


图 13-14　设置显示宽度

步骤 05　设置字段的宽度

设置所有字段固定宽度为 5 厘米，如图 13-15 所示。

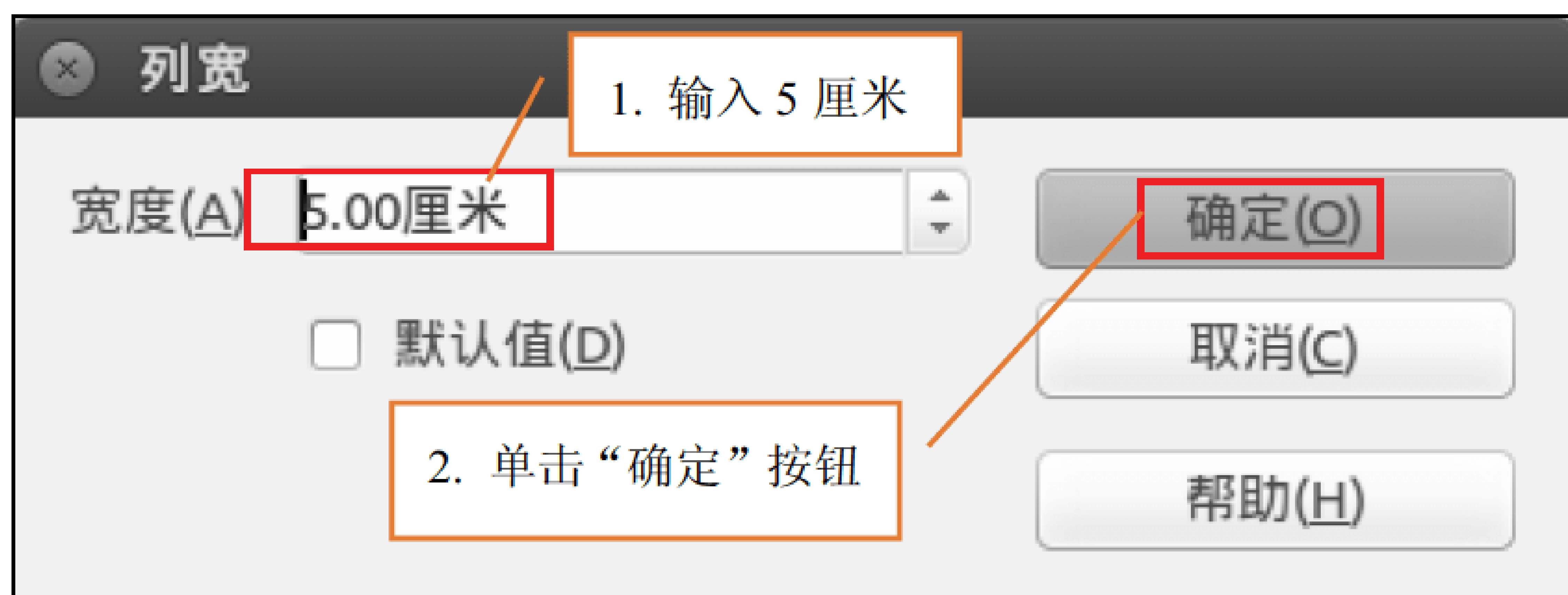


图 13-15　设置字段的宽度

步骤 06　查看 feature 字段

我们先看一下原始数据，思考如何处理这些字段（参考图 13-16 和图 13-17）。各字段的说

明如表 13-2 所示。

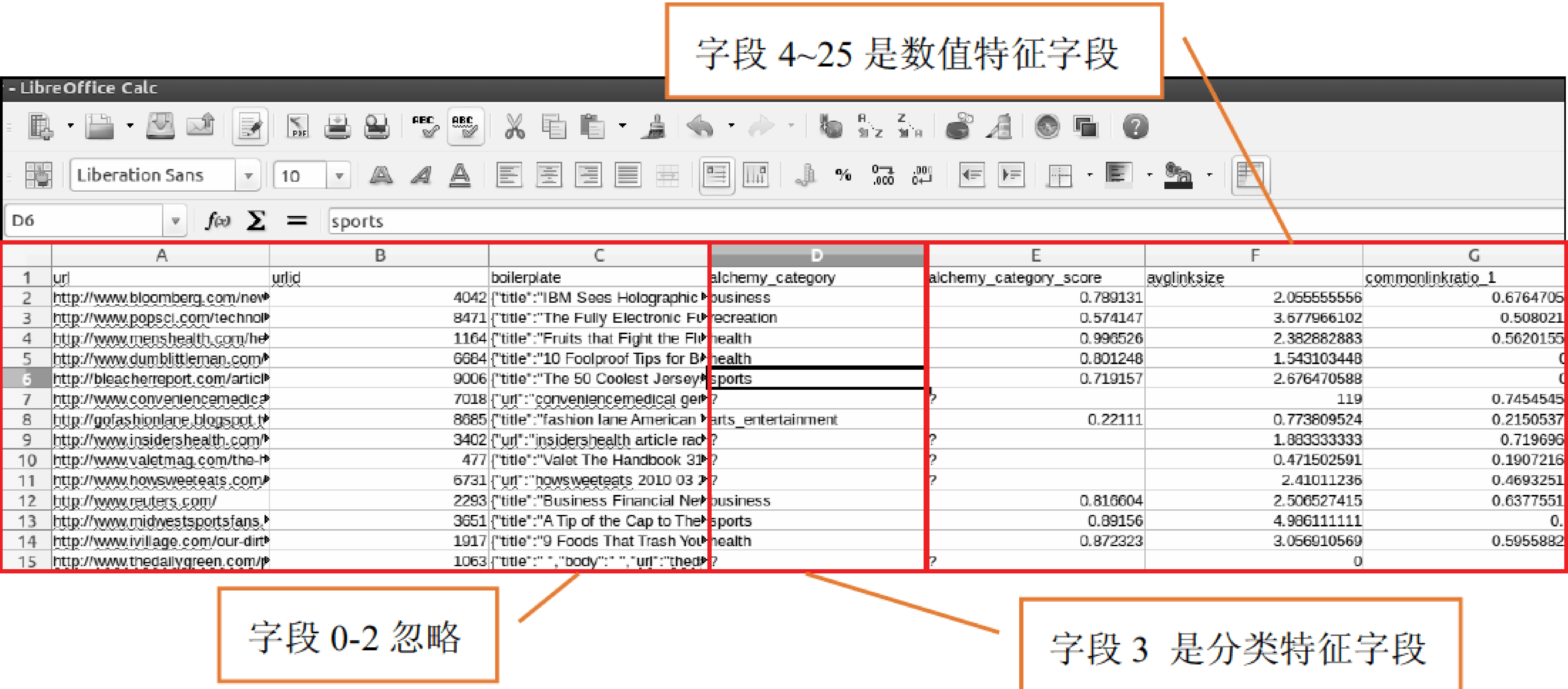


图 13-16　查看 feature 字段（一）

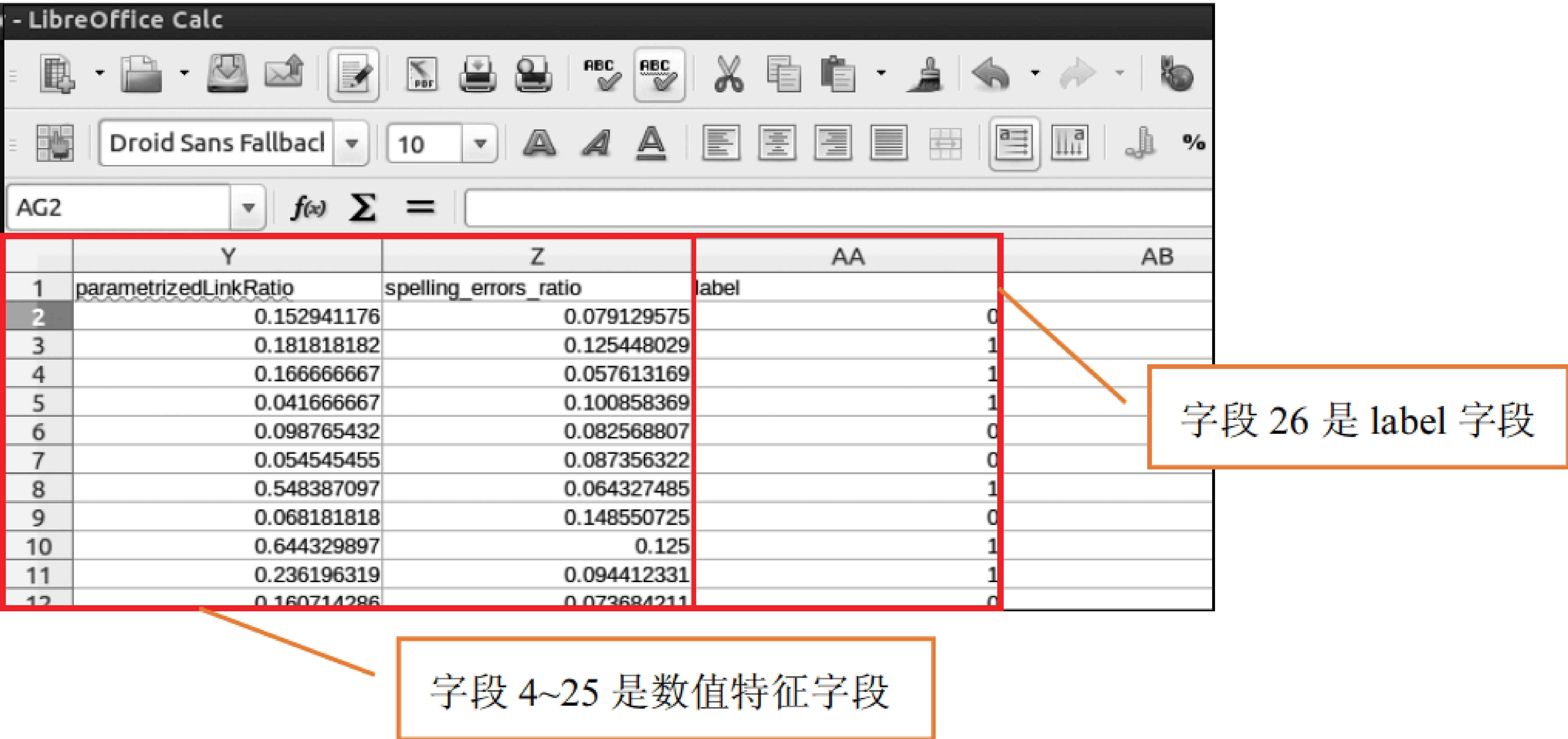


图 13-17　查看 feature 字段（二）

表 13-2　feature 字段说明

字段	字段分类	说明
字段0~2	忽略	url、urlid 对于网站是否长久存在没有太大的关系，所以忽略
字段3	categorical features 分类特征字段	网页分类，例如 business、health、sports……
字段4~25	numerical features 数值特征字段	内容是有关此网页的数值特征，例如链接的数目、影像的比例……
字段26	label 标签字段	label 标签字段，也是数值字段： • 1 代表 evergreen（长青的），此网页会持续让用户感兴趣 • 0 代表 non-evergreen，此网页是暂时性的

步骤 07 用 LibreOffice Calc 电子表格查看 test.tsv

以类似的步骤打开 test.tsv。test.tsv 字段没有 label 字段，其余字段与 train.tsv 完全相同，如图 13-18 所示。

图 13-18 查看 test.tsv 文件

步骤 08 train.tsv 与 test.tsv 比较（见表 13-3）

表 13-3 train.tsv 与 test.tsv 比较

文件	说明
train.tsv	具有 feature（特征字段）与 label（标签字段），将用于训练数据
test.tsv	只有 feature（特征字段），将用于预测数据

13.4.4 复制到项目目录

后续我们将在本地或 cluster 模式运行 Spark，所以我们先将文件复制到本地与 HDFS 目录。

步骤 01 复制下载文件至项目 data 子目录

在“终端”程序中输入下列命令，将我们之前下载的 StumbleUpon 数据集文件复制到项目的 data 子目录。

➢ **复制 train.tsv 到项目的 data 子目录**

```
cp ~/ 下载 /train.tsv ~/pythonwork/PythonProject/data
```

➢ **复制 test.tsv 到项目的 data 子目录**

```
cp ~/ 下载 /test.tsv ~/pythonwork/PythonProject/data
```



➢ **查看项目的 data 子目录**

```
ll ~/pythonwork/PythonProject/data/*.tsv
```

执行后的屏幕显示如图 13-19 所示。

```
hduser@master: ~
hduser@master:~$ cp ~/下载/train.tsv ~/pythonwork/PythonProject/data
hduser@master:~$ cp ~/下载/train.tsv ~/pythonwork/PythonProject/data
hduser@master:~$ ll ~/pythonwork/PythonProject/data/*.tsv
-rwxrwxrwx 1 root root  9428650  2月  2 14:49 /home/hduser/pythonwork/PythonProject/data/test.tsv*
-rwxrwxrwx 1 root root 21972916  3月 24 12:15 /home/hduser/pythonwork/PythonProject/data/train.tsv*
hduser@master:~$
```

图 13-19　复制下载文件至项目 data3 目录

步骤 02 复制到 HDFS（见图 13-20）

➢ **切换到项目的 data 子目录**

```
cd ~/pythonwork/PythonProject/data
```

➢ **复制到 HDFS data 目录**

第 9 章应该已经创建了 HDFS 目录 /user/hduser/data。

```
hadoop fs -copyFromLocal *.tsv /user/hduser/data
```

➢ **HDFS data 目录**

```
hadoop fs -ls /user/hduser/data/*.tsv
```

```
hduser@master: ~/pythonwork/PythonProject/data
hduser@master:~$ cd ~/pythonwork/PythonProject/data
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -copyFromLocal *.tsv /user/hduser/data
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -ls /user/hduser/data/*.tsv
-rw-r--r--   3 hduser supergroup    9428650 2016-03-24 12:29 /user/hduser/data/test.tsv
-rw-r--r--   3 hduser supergroup   21972916 2016-03-24 12:29 /user/hduser/data/train.tsv
```

图 13-20　复制到 HDFS

13.5 使用 IPython Notebook 示范

为了让大家更容易理解决策树算法，我们先使用 IPython Notebook 示范。使用 IPython Notebook 具有互动性的好处，可以看到命令执行后的结果。以下示范在本地执行，读者也可以参考第 9.9 节的说明，在不同的模式运行 IPython Notebook。

在 local 模式启动 IPython Notebook

➢ **切换到 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **在 local 模式运行 IPython Notebook**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook"
MASTER=local[*] pyspark
```

13.6 如何进行数据准备

StumbleUpon 数据集的原始数据是文本文件，我们必须经过一连串的处理，提取特征字段与标签字段，建立训练所需的数据格式 LabeledPoint，并以随机方式按照 8:1:1 比例把数据分割为 3 个部分 trainData、validationData、testData：

- trainData（训练数据）：以此数据训练模型。
- validationData（验证数据）：作为评估模型使用。
- testData（测试数据）：作为测试数据使用。

程序流程示意图如图 13-21 所示。

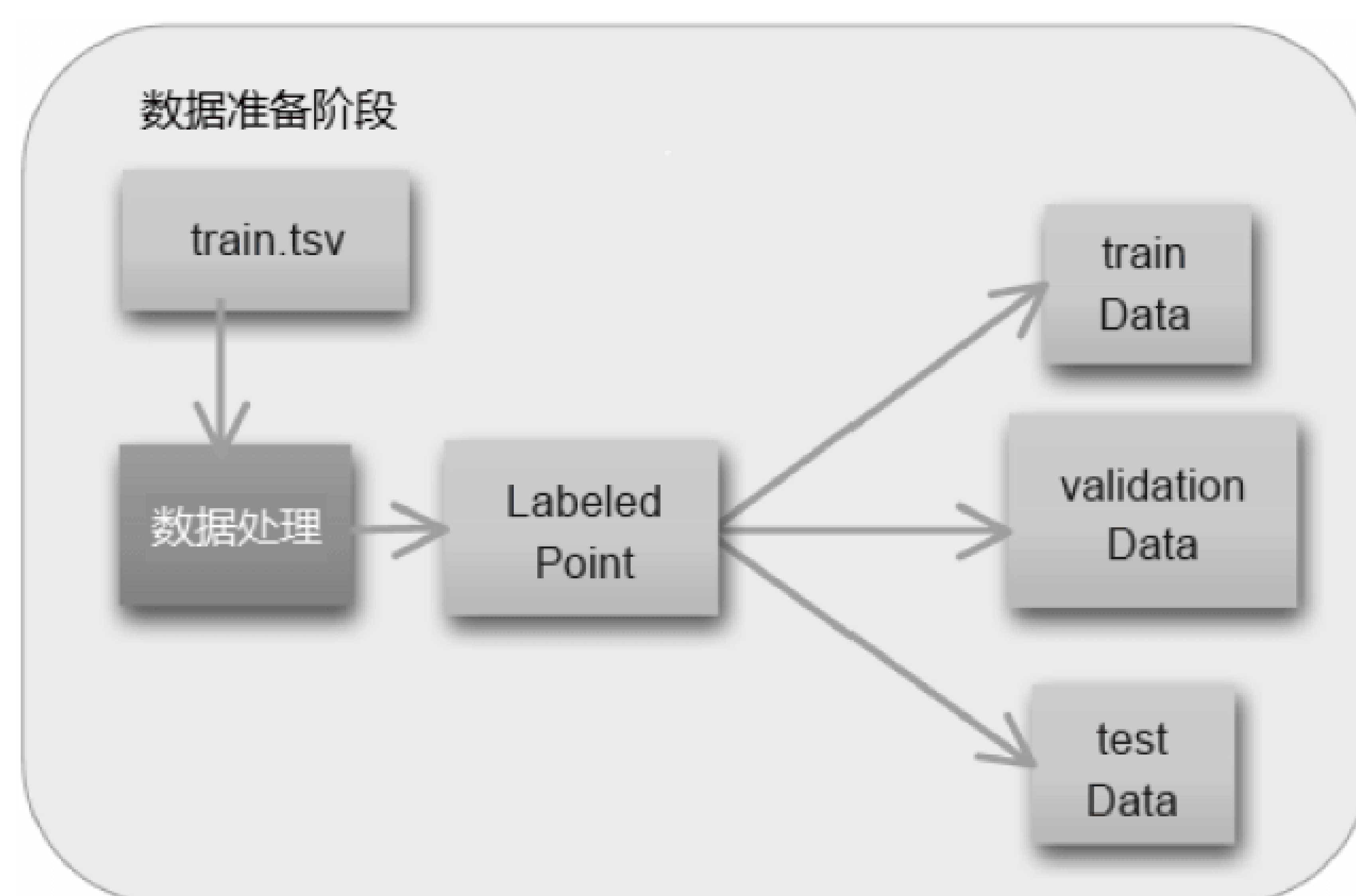


图 13-21　程序流程示意图

13.6.1 导入并转换数据

步骤 01 配置文件读取路径

首先我们必须配置文件读取路径，如图 13-22 所示。

```
In [2]: global Path
        if sc.master[0:5]=="local" :
          Path="file:/home/hduser/pythonwork/PythonProject/"
        else:
          Path="hdfs://master:9000/user/hduser/"
```

图 13-22　配置文件读取路径

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，就代表当前是本地运行，直接读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，就有可能是 YARN Client 或 Spark Stand Alone，必须读取 HDFS 文件。

步骤 02 读取文本文件并查看数据

以 sc.textFile 读取 train.tsv，并使用 take(2) 查看前两项数据，如图 13-23 所示。

```
In [5]: print("开始导入数据 ...")
        rawDataWithHeader = sc.textFile(Path+"data/train.tsv")
        rawDataWithHeader.take(2)
```

图 13-23　读取文本文件

运行结果如图 13-24 所示。

```
开始导入数据 . . .
Out[5]: [u'"url"\t"urlid"\t"boilerplate"\t"alchemy_category"\t"alchemy_category_score"\t"
avglinksize"\t"commonlinkratio_1"\t"commonlinkratio_2"\t"commonlinkratio_3"\t"com
monlinkratio_4"\t"compression_ratio"\t"embed_ratio"\t"framebased"\t"frameTagRatio
"\t"hasDomainLink"\t"html_ratio"\t"image_ratio"\t"is_news"\t"lengthyLinkDomain"\t
"linkwordscore"\t"news_front_page"\t"non_markup_alphanum_characters"\t"numberOfLi
nks"\t"numwords_in_url"\t"parametrizedLinkRatio"\t"spelling_errors_ratio"\t"label
"',
 u'"http://www.bloomberg.com/news/2010-12-23/ibm-predicts-holographic-calls-air-b
reathing-batteries-by-2015.html"\t"4042"\t"{""title"":""IBM Sees Holographic Call
s Air Breathing Batteries ibm sees holographic calls, air-breathing batteries"","
"body"":""A sign stands outside the International Business Machines Corp IBM Alma
den Research Center campus in San Jose California Photographer Tony Avelar Bloomb
erg Buildings stand at the International Business Machines Corp IBM Almaden Resea
rch Center campus in the Santa Teresa Hills of San Jose California Photographer T
```

1. 字段名

2. 数据

图 13-24　查看数据

从中可以发现 train.tsv 有下列几个问题：

（1）第一项数据是字段名。

如图 13-24 所示，第一项数据是字段名而不是数据的必须删除。

（2）每一项数据以“ \t”制表符分隔每一个字段（见图 13-25）。

```
[u'"url"\t"urlid"\t"boilerplate"\t"alchemy_ca
```

“\t”制表符

图 13-25　每一项数据以“\t”分隔字段

（3）每一个字段前后都有双引号“"”分隔（见图 13-26）。

```
[u'"url"\t"urlid"\t"boilerplate"\t"alc
```

双引号“"”分隔

图 13-26　每一个字段由“"”分隔

（4）有些字段无数据，以问号“?”表示。

请参考第 13.2.3 节以 LibreOffice Calc 电子表格查看 train.tsv，如图 13-27 所示。

D	E
alchemy_category	alchemy_category_score
business	0.789131
recreation	0.574147
health	0.996526
health	0.801248
sports	0.719157
?	?
arts_entertainment	0.22111
?	?
?	?

无数据，以问号“?”表示

图 13-27　以“？”表示无数据

步骤 03 读取文本文件

为解决 train.tsv 的上列问题，我们编写图 13-28 所示的程序来处理数据。

```
In [4]: print("开始导入数据 ...")
        rawDataWithHeader = sc.textFile(Path+"data/train.tsv")
        header = rawDataWithHeader.first()
        rawData = rawDataWithHeader.filter(lambda x:x !=header)
        rData=rawData.map(lambda x: x.replace("\"", ""))
        lines = rData.map(lambda x: x.split("\t"))
        print("共计：" + str(lines.count()) + "项")

开始导入数据...
共计：7395项
```

图 13-28　编写程序读取文本文件

以上程序代码说明如下：

➢ **导入 train.tsv**

读取 train.tsv 并载入 rawDataWithHeader。

```
rawDataWithHeader = sc.textFile(Path+"data/train.tsv")
```

➢ **删除第一项字段名**

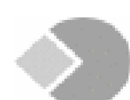
因为 tsv 文件第一项数据是字段名而不是数据，所以必须删除第一项数据。

```
header = rawDataWithHeader.first()
rawData = rawDataWithHeader.filter(lambda x:x !=header)
```

首先使用 .first() 获取第一行表头数据，然后使用 .filter 筛选不是第一项字段的数据。

➢ **删除双引号**

使用 x.replace("\"", "")将双引号“"”删除。



```
rData=rawData.map(lambda x: x.replace("\"", ""))
```

➤ **获取每一行数据字段**

因为文本文件是以 Tab 键分隔字段的，所以使用 "\t" 分隔字段来获取每一个字段。

```
lines = rData.map(lambda x: x.split("\t"))
```

➤ **显示数据项数**

```
print(" 共计: " + str(lines.count()) + " 项 ")
```

步骤 04 查看第一项数据

字段 0~2（网址、网址 ID、模板文字）与判断网页是暂时性的（ephemeral）或是长青的（evergreen）关系不大，所以会忽略掉。以下数据 lines.first() 读取第一项数据，[3:] 读取第 3 个字段之后的数据。

```
In [5]: lines.first()[3:]
```

运行后结果如图 13-29 所示。

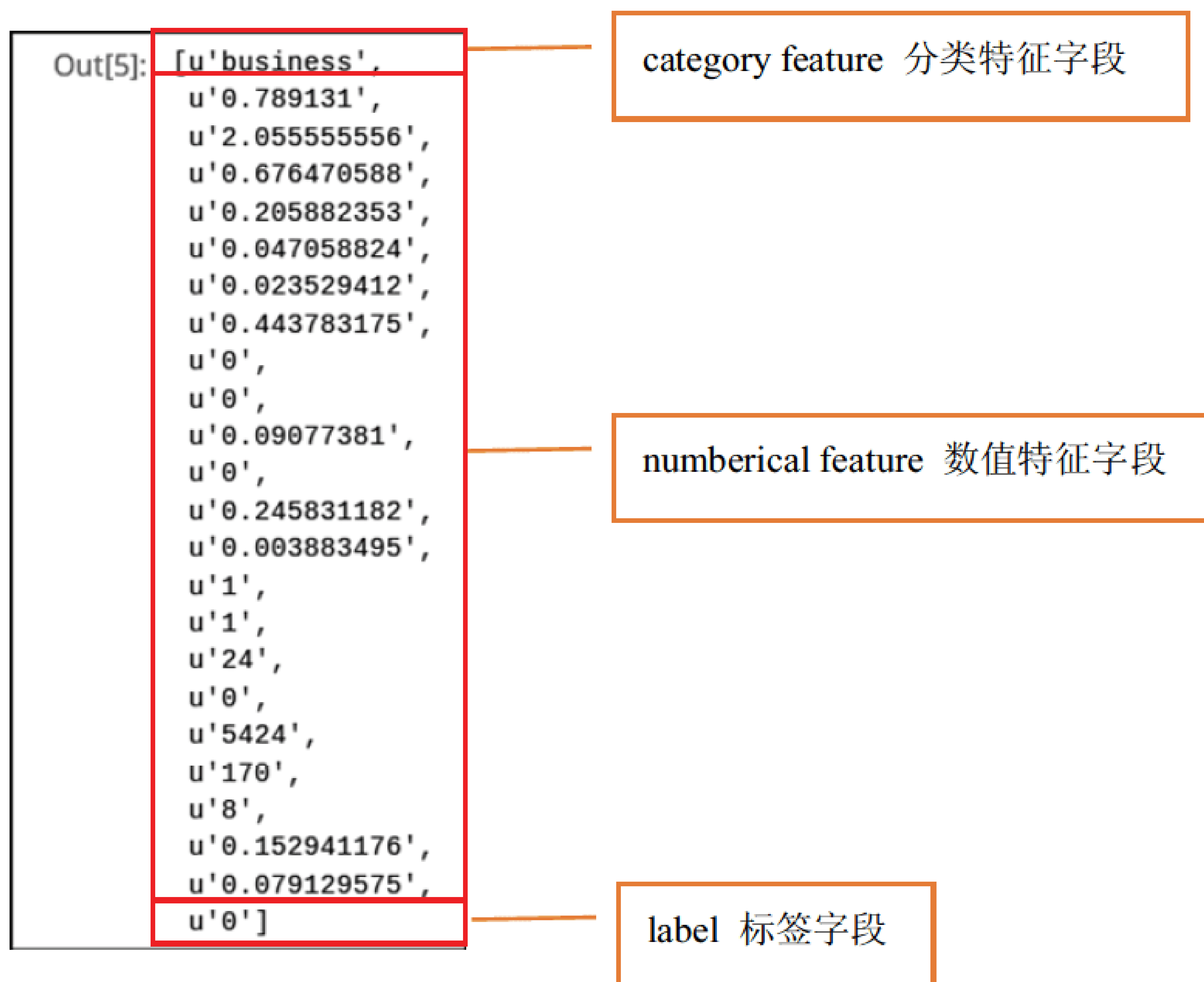


图 13-29　查看数据

13.6.2　提取 feature 特征字段

➤ **编写 extract_features 函数**

为了提取数据的 feature 特征字段，可以编写 extract_features 函数，如图 13-30 所示。

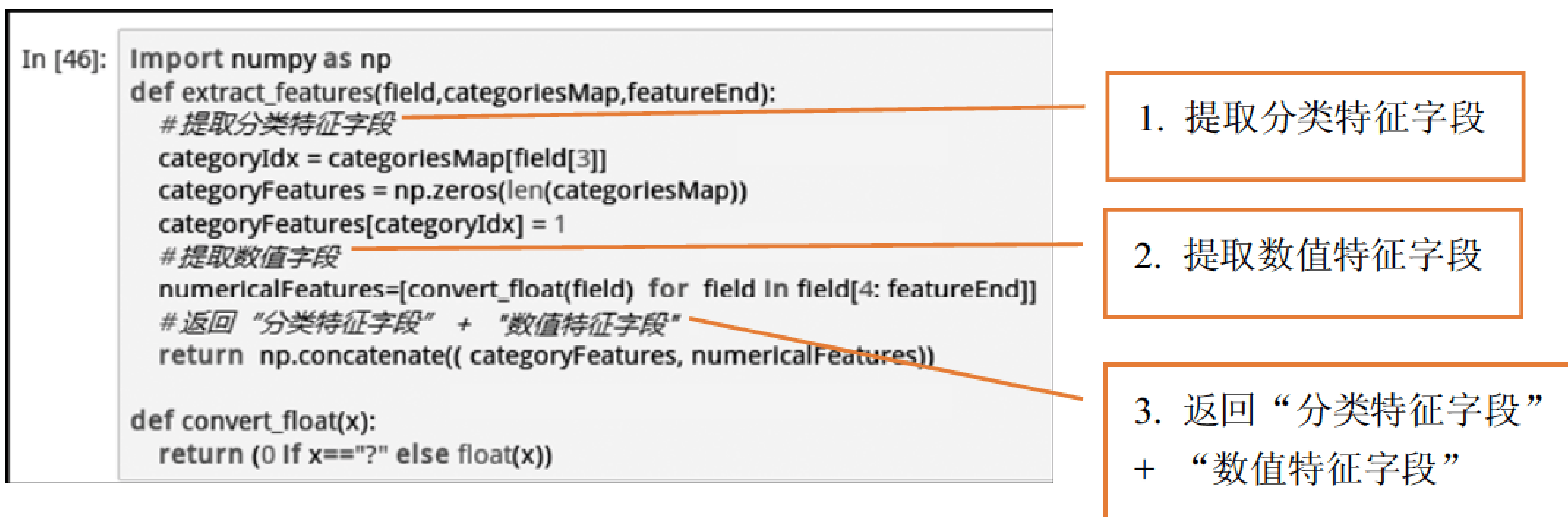


图 13-30 编写 extract_features 函数

以上程序代码说明如表 13-4 所示。

表 13-4 extract_features 函数代码说明

命令	详细说明
import numpy as np	导入 numpy 模块
def extract_features(field, categoriesMap, featureEnd):	定义 extract_features，传入下列参数： • field：每一项数据 • categoriesMap：字典 • featureEnd：提取特征字段结尾

extract_features 函数主要分为以下 3 部分。

（1）提取分类特征字段：提取结果存放在 categoryFeatures List 中，在第 13.6.3 小节会详细说明。

（2）提取数值特征字段：提取结果存放在 numericalFeatures List 中，在第 13.6.4 小节会详细说明。

（3）返回“分类特征字段”＋“数值特征字段”：在第 13.6.5 小节会详细说明。

13.6.3 提取分类特征字段

此数据集的第 3 个字段是 alchemy_category 网页分类，以上程序代码主要是处理字段 3 网页分类，这是分类特征（Category Feature）字段，必须转换为数值字段才能够被分类算法使用。转换的方法是以 OneHotEncoder 的方式进行。如果网页分类有 N 个分类，就会转换为 N 个数值字段。处理步骤如下：

（1）创建 categoriesMap 网页分类字典，一个分类对应一个数字。

（2）分类特征字段使用 categoriesMap（网页分类字典），转换为数字，例如 business 通过 categoriesMap 转换后 categoriesIdx=2。

（3）categoriesIdx= 2 再以 OneHotEncoder 方式转换为 categoryFeatures List，从 0 算起，位置 2 是 1，结果是 0,0,1,0,0,0,0,0,0,0,0,0,0,0，共 14 个数值字段。

程序流程示意图如图 13-31 所示。



图 13-31　程序流程示意图

步骤01 建立 categoriesMap 网页分类字典

我们可使用下列指令创建网页分类字典。

```
In [6]: categoriesMap =lines.map(lambda fields: fields[3]) \
           .distinct().zipWithIndex().collectAsMap()
```

以上命令的详细说明如表 13-5 所示。

表 13-5　网页分类字典的命令说明

指令	详细说明
categoriesMap =lines.map	将之前读取的文字数据经过一连串的处理后存入 categoriesMap
map(lambda fields: fields[3])	读取第 3 个字段
.distinct()	保留不重复的数据
.zipWithIndex()	将第 3 个字段中不重复的数据进行编号
.collectAsMap()	转换为 dict 字典格式

步骤02 查看 categoriesMap 字典

创建完成后，查看 categoriesMap 字典（见图 13-32）。

```
In [43]: categoriesMap

Out[43]: {u'?': 12,
          u'arts_entertainment': 13,
          u'business': 2,
          u'computer_internet': 3,
          u'culture_politics': 5,
          u'gaming': 0,
          u'health': 11,
          u'law_crime': 7,
          u'recreation': 1,
          u'religion': 9,
          u'science_technology': 6,
          u'sports': 8,
          u'unknown': 4,
          u'weather': 10}
```

图 13-32　查看 categoriesMap 字典

我们可以看到每一个分类都对应一个数字，例如 business 对应 2。

步骤03 查看 categoriesMap 项数

我们还可以使用 len() 函数来查看 categoriesMap 项数，显示共有 14 项数据。

```
In [44]: len(categoriesMap)
Out[44]: 14
```

步骤04 查看 categoriesMap 数据类型

可以用type() 函数来查看categoriesMap 数据类型是dict 字典。

```
In [45]: type(categoriesMap)
Out[45]: dict
```

步骤05 提取分类特征字段

提取分类特征字段，程序代码如图 13-33 所示。

```
#提取分类特征字段
categoryIdx = categoriesMap[field[3]]
categoryFeatures = np.zeros(len(categoriesMap))
categoryFeatures[categoryIdx] = 1
```

图 13-33 提取分类特征字段

➢ **网页分类转换为数值**

```
categoryIdx = categoriesMap[field[3]]
```

网页分类原本是文字，我们必须转为数值。使用 categoriesMap 字典传入字段 3，并且转换为数值。例如，此项数据分类是 business，我们会转换为 categoryIdx=2。

➢ **初始化 categoryFeatures**

```
categoryFeatures = np.zeros(len(categoriesMap))
```

上述语句我们使用 np.zeros，传入参数 len(categoriesMap)，也就是 14。命令执行后会创建一个新的 categoryFeatures List，其中的值都是 0，List 的大小是 14，也就是 0,0,0,0,0,0,0,0,0,0,0,0,0,0。

➢ **设置 List 相对应的位置是 1**

```
categoryFeatures[categoryIdx] = 1
```

设置 categoryFeatures List 相对应的位置是 1，其余位置默认是 0。List 的位置是从 0 开始的。若字段 3 的值是 business，则经过字典转换后 categoryIdx=2。所以我们会设置 categoryFeatures List 中从 0 算起第 2 个位置是 1，结果是 0,0,1,0,0,0,0,0,0,0,0,0,0,0。

13.6.4　提取数值特征字段

此数据集的第 4 至 25 字段（也就是倒数第二个字段）是数值特征字段，必须转化为数值。

步骤01 定义 convert_float

因为文本文件中很多字段数据没有数值，会以“？”代表，所以下列程序会判断是否为“？”。如果是，就返回数值 0；如果不是 0，就转换为 float。

```
def convert_float(x):
    return (0 if x=="?" else float(x))
```

步骤02 处理数值字段

处理数值字段的程序代码如下：

```
numericalFeatures=[convert_float(field) for field in field[4: featureEnd]]
```

（1）因为我们是读取文本文件，数据类型是 string，所以每一个字段必须转换为 float。

（2）[convert_float(field) for field in record[4: featureEnd]]会自第 4 字段到 featureEnd 字段，执行 convert_float 函数，转换为 float。

（3）以上 featureEnd 是参数，调用 extract_features 时会传入参数 len(r) – 1，也就是倒数第 2 个字段。

（4）将转换后的结果存入 numericalFeatures 并返回。

13.6.5　返回特征字段

经过上列处理后会产生 numericalFeatures 与 categoryFeatures，我们可以使用 np.concatenate 将 List 相加并返回。

```
return np.concatenate(( categoryFeatures, numericalFeatures))
```

13.6.6　提取 label 标签字段

下面编写 extract_label 函数，提取 label 字段。

```
In [16]: def extract_label(field):
             label=(field[-1])
             return float(label)
```

以上函数传入 field 参数是单项数据，field[-1] 获取最后一个字段，也就是 label 字段，最后返回 float(label) 转换为 float 之后的 label。

13.6.7 建立训练评估所需的数据

后续进行 Decision Tree 的训练必须提供 LabeledPoint 格式的数据，所以我们必须先建立 LabeledPoint 数据。

步骤01 创建 LabeledPoint 数据

LabeledPoint 是由 label 与 feature 组成的。之前的步骤我们已编写了 extract_label() 与 extract_features() 函数，接下来我们要使用这两个函数建立训练评估所需的 LabeledPoint。

```
In [50]: from pyspark.mllib.regression import LabeledPoint
         labelpointRDD = lines.map( lambda r:
             LabeledPoint(
                     extract_label(r),
                     extract_features(r,categoriesMap,len(r) - 1)))
```

以上程序代码的详细说明如表 13-6 所示。

表 13-6 程序代码说明

程序代码	说明
from pyspark.mllib.regression import LabeledPoint	导入 LabeledPoint 模块
labelpointRDD = lines.map(lambda r:	针对 lines 每一项数据转换，并以匿名函数传入参数 r，转换后将结果存入 labelpointRDD
LabeledPoint(	建立 LabeledPoint 数据
extract_label(r)	提取最后一个字段作为 label 字段
extract_features(r, categoriesMap, len(r) – 1)))	extract_features 提取特征字段，传入参数： • r：传入的每一项数据 • categoriesMap：网页分类字典 • featureEnd：设置为 len(r) – 1，因为最后一个字段是 label

步骤02 查看第一项未处理前数据

我们可以查看 lines 第一项未处理前的数据，如图 13-34 所示。

```
In [82]: print lines.first()[3:]
         [u'business', u'0.789131', u'2.055555556', u'0.676470588', u'0.205882353', u'0.0470588
         24', u'0.023529412', u'0.443783175', u'0', u'0', u'0.09077381', u'0', u'0.245831182',
         u'0.003883495', u'1', u'1', u'24', u'0', u'5424', u'170', u'8', u'0.152941176', u'0.07
         9129575', u'0']
```

分类特征字段 business

label 是 0

图 13-34 查看第一项未处理前的数据

以上执行结果，我们是读取 [3:]，即读取自第 3 字段之后的数据，第一个字段是分类特



征字段 business，其余字段是数值特征字段，最后是 label。

步骤 03　查看 LabelPoint 第一项数据

lines 数据经过 extract_label 与 extract_features 处理后产生 LabeledPoint。我们可以查看 LabeledPoint 第一项数据，如图 13-35 所示。

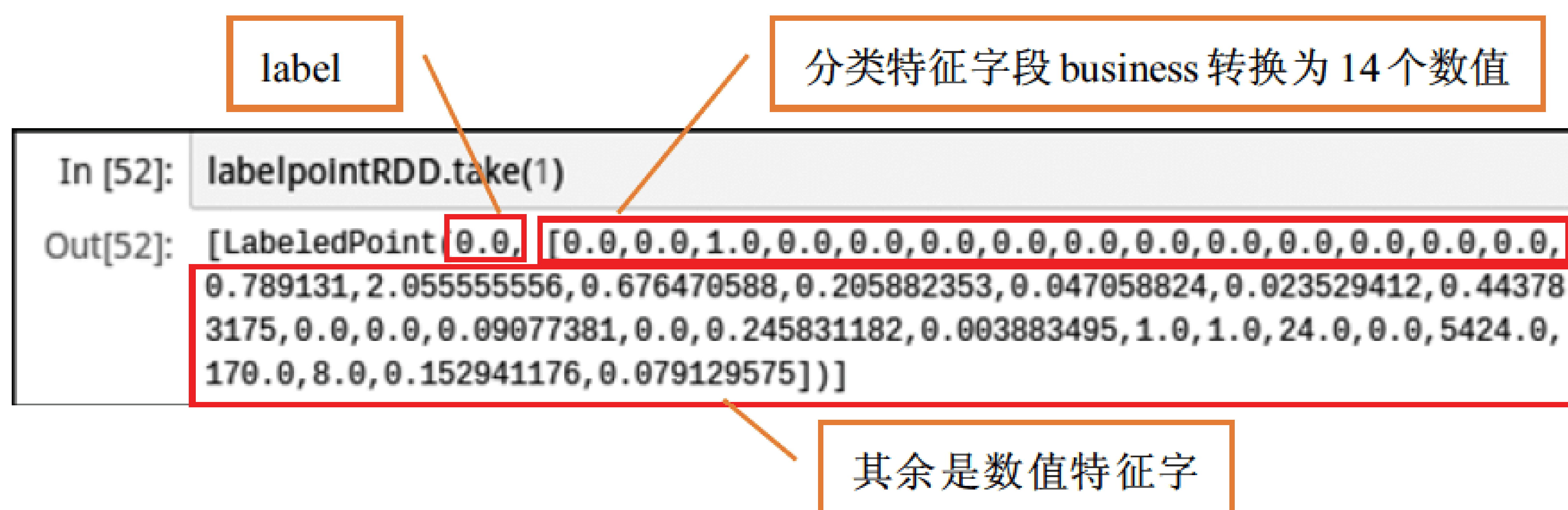


图 13-35　查看 LabeledPoint 第一项数据

我们可以看到最前面的数字是 label，然后是分类特征字段转换为 14 个数值字段，其余是数值特征字段。

13.6.8　以随机方式将数据分为 3 部分并返回

将之前所建立的 RDD[LabeledPoint] 数据以 randomSplit 随机方式按照 8:1:1 的比例分隔为 3 部分：trainData（训练数据）、validationData（验证数据）、testData（测试数据），然后用 print 显示 trainData、validationData、testData 项数，如图 13-36 所示。

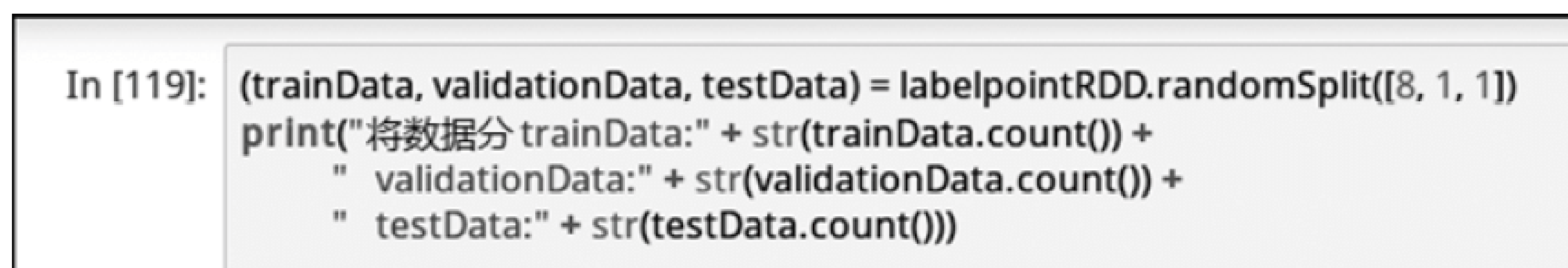


图 13-36　以随机方式将数据分为 3 部分

13.6.9　编写 PrepareData(sc) 函数

为了方便后续进行数据处理，我们将之前步骤中数据处理的命令全部收集在 PrepareData(sc) 函数中。

步骤 01　编写 PrepareData(sc) 函数（见图 13-37）

```
In [204]: def PrepareData(sc):
    #---------------------1. 导入并转换数据-------------
    global Path
    if sc.master[0:5]=="local" :
      Path="file:/home/hduser/pythonwork/PythonProject/"
    else:
      Path="hdfs://master:9000/user/hduser/"

    print(" 开始导入数据...")
    rawDataWithHeader = sc.textFile(Path+"data/train.tsv")
    header = rawDataWithHeader.first()
    rawData = rawDataWithHeader.filter(lambda x:x !=header)
    rData=rawData.map(lambda x: x.replace("\"", ""))
    lines = rData.map(lambda x: x.split("\t"))
    print("共计：" + str(lines.count()) + "项")
    #--------------------- 2.建立训练评估所需数据 RDD[LabeledPoint]-------------
    categoriesMap = lines.map(lambda fields: fields[3]). \
                        distinct().zipWithIndex().collectAsMap()
    labelpointRDD = lines.map( lambda r:LabeledPoint(
                    extract_label(r),
                    extract_features(r,categoriesMap ,-1)
                    ))
    #---------------------3.以随机方式将数据分为3个部分并返回 -------------
    (trainData, validationData, testData) = labelpointRDD.randomSplit([8, 1, 1])
    print("将数据分trainData:" + str(trainData.count()) +
          "   validationData:" + str(validationData.count()) +
          "   testData:" + str(testData.count()))
    return (trainData, validationData, testData, categoriesMap) #返回数据
```

图 13-37　编写 PrepareData(sc)函数

步骤 02　执行 PrepareData(sc) 函数

执行 PrepareData(sc) 函数，如图 13-38 所示。

```
In [159]: (trainData, validationData, testData, categoriesMap) =PrepareData(sc)

          开始导入数据...
          共计：7395 项
          将数据分trainData:5893   validationData:744   testData:758
```

图 13-38　执行 PrepareData(sc) 函数

步骤 03　将数据暂存在内存中

为了加快后续程序的运行效率，以下列程序代码将 trainData、validationData 暂存在内存中：

```
In [19]: trainData.persist()
         validationData.persist()
         testData.persist()

Out[19]: PythonRDD[33] at RDD at PythonRDD.scala:48
```

13.7 如何训练模型

建立训练所需的数据格式 LabeledPoint 后建立 trainData，我们将使用 trainData 执行

DecisionTree 训练并建立模型。

执行 DecisionTree 训练的程序代码如下：

```
In [20]: from pyspark.mllib.tree import DecisionTree
         model=DecisionTree.trainClassifier( \
             trainData, numClasses=2, categoricalFeaturesInfo={}, \
             impurity="entropy", maxDepth=5, maxBins=5)
```

上述程序代码首先导入 DecisionTree 模块，然后执行 DecisionTree.trainClassifier 进行训练。各模块介绍如表 13-7 所示。

```
DecisionTree.trainClassifier(input, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins)
```

返回：DecisionTreeModel。

表 13-7　参数说明

参数	说明
input	输入的训练数据
numClasses	分类数目
categoricalFeaturesInfo	设置分类特征字段信息
impurity	决策树的 impurity 评估方法有两种方式：gini 基尼系数、entropy 熵
maxDepth	决策树最大深度
maxBins	决策树每一个节点最大分支数

其中，categoricalFeaturesInfo 参数是设置分类特征字段的信息，在本范例中采用 OneHotEncoding 转换分类特征字段，所以设置为空的 dict {}。

13.8　如何使用模型进行预测

建立 DecisionTree 模型后，我们可以使用此模型预测 test.tsv 数据。test.tsv 只有 feature（特征字段），我们将使用此特征字段预测网页是暂时性的（ephemeral ）或是长青的（evergreen）。程序流程示意图如图 13-39 所示。

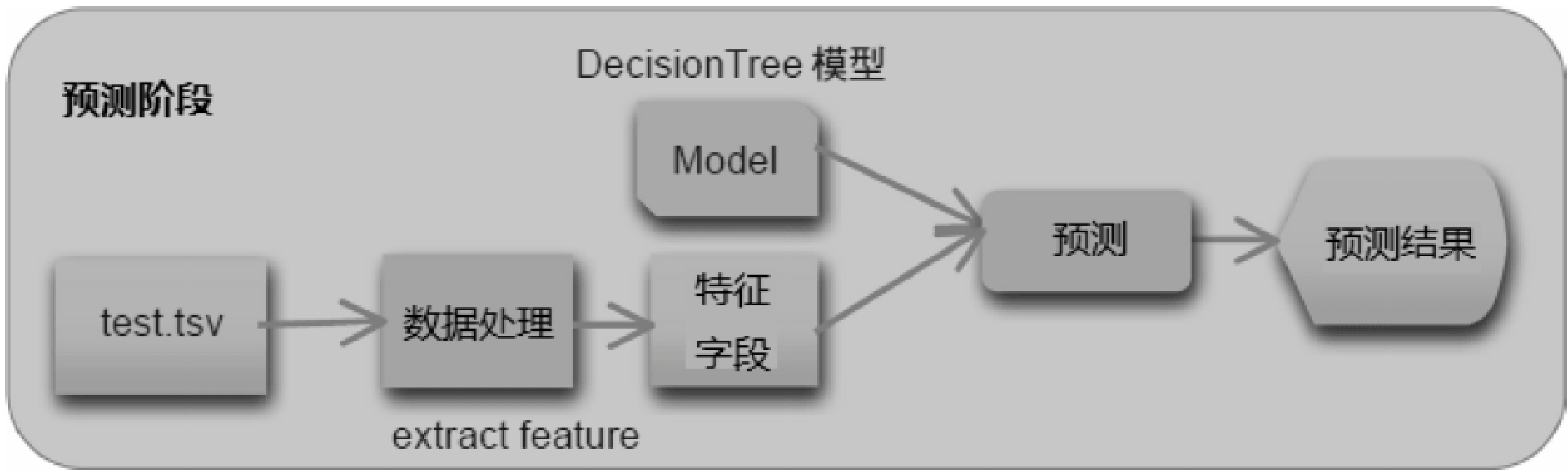


图 13-39　程序流程示意图

步骤01 定义 PredictData 函数进行预测

定义 PredictData 函数，如图 13-40 所示。

```
In [208]: def PredictData(sc,model,categoriesMap):
              print("开始导入数据...")
              rawDataWithHeader = sc.textFile(Path+"data/test.tsv")
              header = rawDataWithHeader.first()
              rawData = rawDataWithHeader.filter(lambda x:x !=header)
              rData=rawData.map(lambda x: x.replace("\"", ""))
              lines = rData.map(lambda x: x.split("\t"))
              print("共计：" + str(lines.count()) + "项")
              dataRDD = lines.map(lambda r:  ( r[0] ,
                              extract_features(r,categoriesMap,len(r))))
              DescDict = {
                   0: "暂时性网页(ephemeral)",
                   1: "长青网页(evergreen)"
               }
              for data in dataRDD.take(10):
                  predictResult = model.predict(data[1])
                  print " 网址：  " +str(data[0])+"\n" +\
                        "          ==>预测:"+ str(predictResult)+ \
                        " 说明 :"+DescDict[predictResult] +"\n"
```

图 13-40 定义 PredictData 函数

以上程序代码与第 13.6.9 小节的 PrepareDate() 类似。下面仅说明差异之处。

➢ 定义 PredictData 函数

程序代码	说明
def PredictData(sc, model, categoriesMap):	定义 PredictData 函数，传入参数： • sc：SparkContext • model：之前训练完成的模型 • categoriesMap：之前建立的网页分类字典

➢ 编写 dataRDD

因为我们希望能显示网页网址与预测网页的结果，所以编写 dataRDD 由网页网址与特征字段组成。

程序代码	说明
dataRDD = lines.map(lambda r: (r[0] , extract_features(r, categoriesMap,len(r))))	编写 dataRDD • 第 0 个字段：r[0] 也是网址 • 第 1 个字段：extract_features 提取特征字段，在此我们传入参数 r、categoriesMap 与 len(r)

以上我们传入参数 len(r) 而不是 len(r)-1，这是因为 test.tsv 只有 feature 字段，没有 label 字段，所以传入参数 len(r)，读取到最后字段。

➢ 使用 dataRDD 进行预测并显示结果

之前步骤产生的 dataRDD 第 0 个字段是网址、第 1 个字段是所有特征字段，用于显示

网址与预测结果。

程序代码	说明

DescDict = {
0: " 暂时性网页 (ephemeral)",
1: " 长青网页 (evergreen)"
}
	原本 label 的值是 0, 1，后续我们希望能在程序中显示 0 和 1 所代表的意义。所以我们建立 DescDict 字典，后续可用此字典转换预测结果

for data in dataRDD.take(10):	读取前 10 项数据
predictResult = model.predict(data[1])	使用 model.predict 进行预测，传入 data[1] 也就是 feature 特征字段。预测结果存入 predictResult
print " 网址： " + str(data[0])	显示 data[0] 网址
预测 :"+ str(predictResult)	显示预测结果
" 说明 :" +DescDict[predictResult]	使用 DescDict 字典转换预测结果说明

步骤 02 定义 PredictData 函数进行预测

编写 PredictData 函数后，我们就用此函数进行预测。下面执行 PredictData 传入 sc, model, categoriesMap 进行预测，如图 13-41 所示。

```
In [22]: print("========== 预测数据 ===============")
         PredictData(sc, model, categoriesMap)
```

图 13-41　定义 PredictData 函数进行预测

预测结果如图 13-42 所示。



图 13-42　预测结果

从图 13-42 可以看到网页网址与预测的结果，我们也可以单击选中网址，查看一下预测结果是否合理。不过我们不太可能用人来判断决策树模型的准确率，必须有一个科学的方法评估模型的准确率，后续章节会介绍。

13.9 如何评估模型的准确率

有了模型之后，我们希望知道这个模型预测的准确率，必须要有一个标准来评估模型的准确率。

13.9.1 使用 AUC 评估二元分类模型

针对二元分类算法，主要是以 AUC（Area under the Curve of ROC）来评估数据模型的好坏。

		Actual 真实值	
		positives	negatives
Predict 预测值	positives	True Positives (TP)	False Positives (FP)
	negatives	False Negatives (FN)	True Negatives (TN)

例如：

- 0 代表暂时性网页。
- 1 代表长青网页。
- 真阳性 True Positives（TP）：预测为 1，实际上为 1。
- 伪阳性 False Positives（FP）：预测为 1，实际上为 0。
- 真阴性 True Negatives（TN）：预测为 0，实际上为 1。
- 伪阴性 False Negatives（FN）：预测为 0，实际上为 0。
- TPR：在所有实际为 1 的样本中被正确地判断为 1 的比例。

  TPR=TP/(TP+FN)
- FPR：在所有实际为 0 的样本中被错误地判断为 1 的比例。

  FPR=FP/(FP+TN)

有了 TPR、FPR 就可以绘出 ROC 曲线图，如图 13-43 所示。AUC（Area under the Curve of ROC）就是 ROC 曲线下的面积。

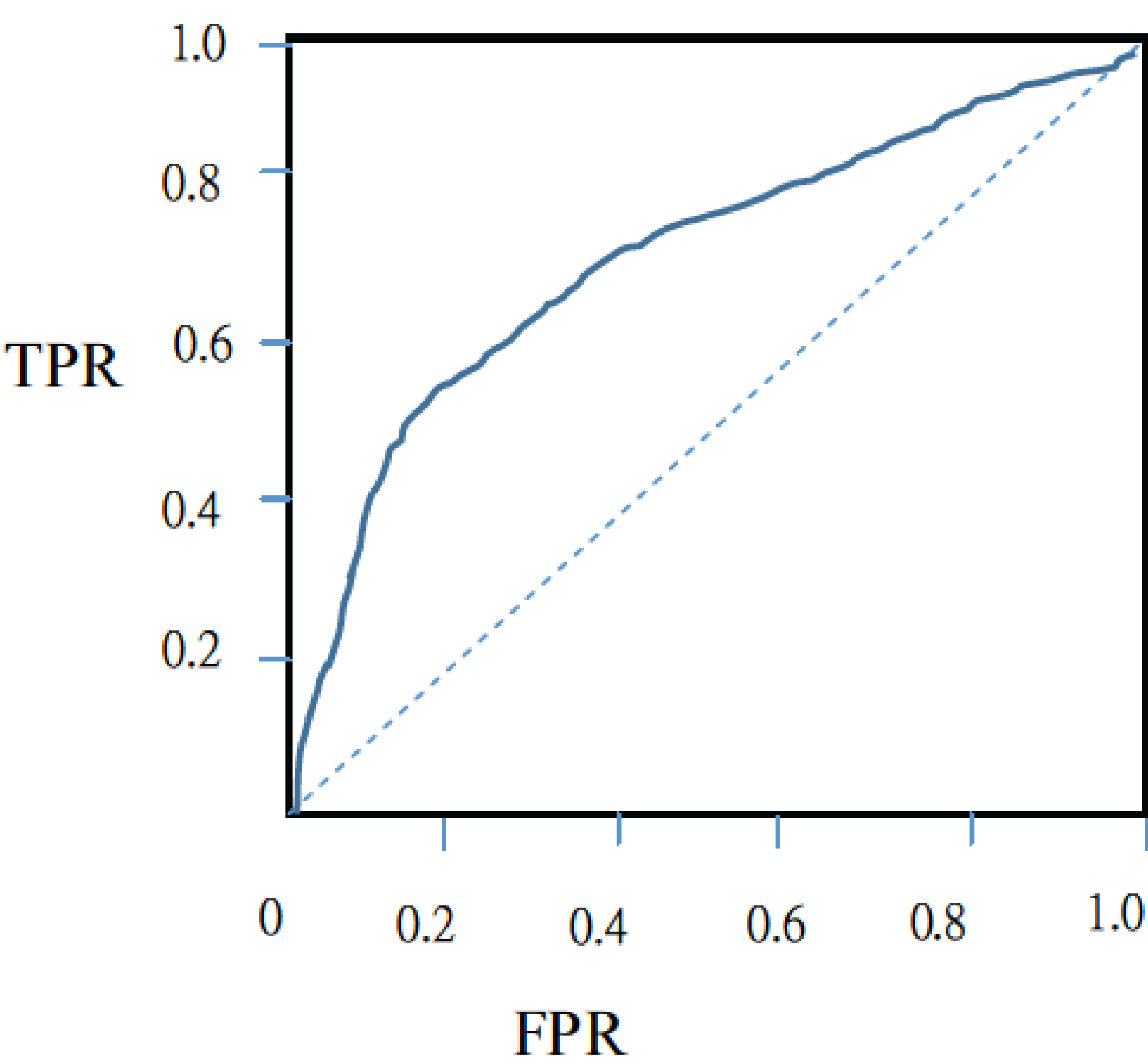


图 13-43　ROC 曲线图

可从 AUC 判断二元分类的优劣，如表 13-8 所示。

表 13-8　从 AUC 判断二元分类的优劣

条件	说明
AUC = 1	最完美的情况，预测准确率 100%，但是不可能存在
0.5 < AUC < 1	优于随机猜测，具有预测价值
AUC = 0.5	与随机猜测一样，没有预测价值
AUC < 0.5	比随机猜测还差，但如果反向预测，就优于随机猜测

13.9.2　计算 AUC

以上说明已经了解 AUC 的概念，接下来要在程序中使用 AUC 评估二元分类模型的好坏。计算 AUC 有些复杂，不过不用担心，MLlib 已经有了这个功能。MLlib 提供了使用 BinaryClassificationMetrics 计算 AUC 的方法，计算的步骤如下：

（1）建立 scoreAndLabels。

（2）使用 scoreAndLabels 建立 BinaryClassificationMetrics，并使用 BinaryClassification Metrics。

步骤 01 创建 scoreAndLabels

建立 scoreAndLabels，如图 13-44 所示。从执行结果中我们可以看到显示了 5 项数据，每一项都是由 score 与 label 组成的。

```
In [21]: score = model.predict(validationData.map(lambda p: p.features))
         scoreAndLabels=score .zip(validationData.map(lambda p: p.label))
         scoreAndLabels.take(5)

Out[21]: [(1.0, 0.0), (0.0, 0.0), (1.0, 1.0), (0.0, 0.0), (0.0, 1.0)]
```

score label

图 13-44 建立 scoreAndLabels

以上程序代码的说明如下：

（1）使用 model.predict 进行预测，传入参数 validationData.map(lambda p: p.features)，也就是 validationData 的 features 部分进行预测，预测结果存在 score 中。

（2）建立 scoreAndLabels: score（预测的结果）使用 .zip 方法结合。validationData.map (lambda p: p.label)验证数据的 label 标签字段。

（3）scoreAndLabels.take(5) 显示前 5 项数据。

步骤 02 编写 BinaryClassificationMetrics 计算 AUC

使用 BinaryClassificationMetrics 计算 AUC。

```
In [25]: from pyspark.mllib.evaluation import BinaryClassificationMetrics
         metrics = BinaryClassificationMetrics(scoreAndLabels)
         print "AUC="+str(metrics.areaUnderROC)

AUC=0.656874990538
```

程序详细说明如下：

（1）导入 BinaryClassificationMetrics 模块。

（2）使用 BinaryClassificationMetrics 传入参数 scoreAndLabels 建立二元分类 metrics。

（3）有了 metrics，我们可以用 areaUnderROC 方法计算 AUC。

步骤 03 编写 evaluateModel 函数

因为后续我们评估模型的机会很多，所以现在编写 evaluateModel 函数，以方便我们后续重复使用它来评估模型。

```
In [24]: def evaluateModel(model, validationData):
             score = model.predict(validationData.map(lambda p: p.features))
             scoreAndLabels=score.zip(validationData.map(lambda p: p.label))
             metrics = BinaryClassificationMetrics(scoreAndLabels)
             AUC=metrics.areaUnderROC
             return( AUC)
```

以上使用 def evaluateModel(model, validationData) 定义函数，传入参数训练完成的 model 模型与 validationData 验证数据，可以帮助我们计算 AUC。

步骤04 执行 evaluateModel 函数

以下程序代码执行 evaluateModel 函数，输入已训练完成的 model 模型与 validationData 验证数据，执行结果 AUC 大约为 0.65。

```
In [26]: AUC=evaluateModel(model, validationData)
         print "AUC="+str(AUC)
```

```
AUC=0.656874990538
```

13.10 模型的训练参数如何影响准确率

在训练模型时我们会输入不同的参数。其中，DecisionTree 参数 impurity、maxDepth、maxBins 的值会影响准确率以及训练所需的时间。我们将以图表显示这些参数值、准确率与训练所需的时间。

我们每次只会评估单个参数的不同数值，例如评估 maxDepth 参数的不同数值 [3, 5, 10, 15, 20, 25] ，执行的步骤如下：

（1）用 DecisionTree.trainClassifier 进行训练传入 trainData 与单个参数的不同数值。

（2）建立模型后，用 validationData 评估模型的 AUC 准确率。

（3）训练与评估模型重复执行多次， 产生多项的 AUC 与运行时间，并存储于 metrics RDD 中。

（4）全部执行完成后，将 metrics RDD 转换为 Pandas DataFrame。

（5）Pandas DataFrame 可绘制 AUC 与运行时间图表，用于显示不同参数的准确率与执行时间的关系。

处理流程的示意图如图 13-45 所示。

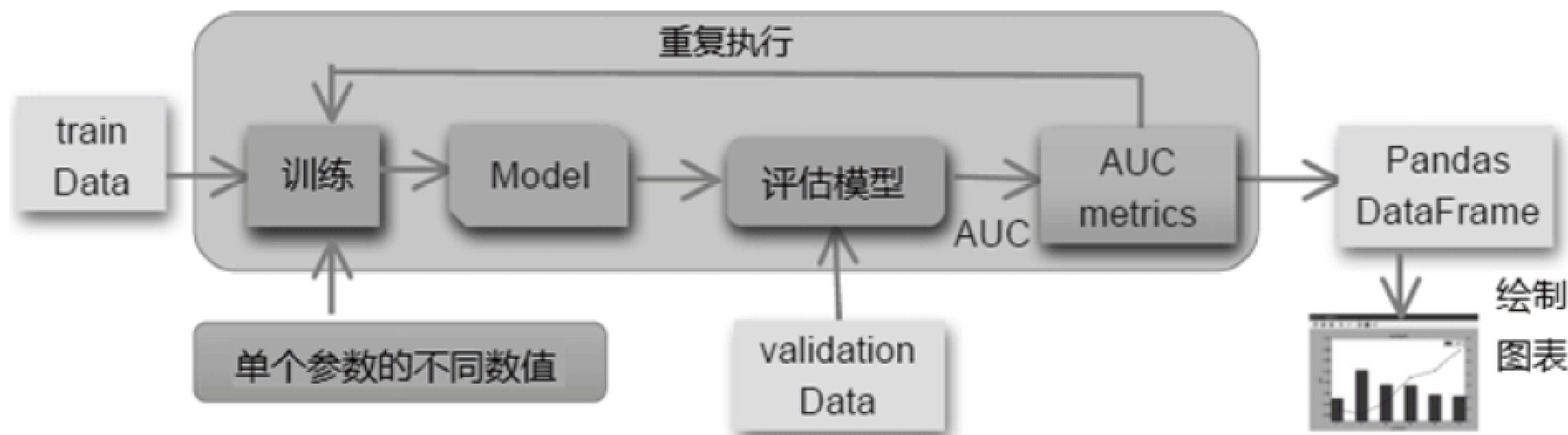


图 13-45　处理流程的示意图

13.10.1　建立 trainEvaluateModel

我们将通过一个函数来建立 trainEvaluateModel，其中包含训练与评估的功能，并且计算

训练评估所需的时间。

步骤01 建立 trainEvaluateModel

编写 trainEvaluateModel 函数，如图 13-46 所示。

```
In [41]: from time import time
         def trainEvaluateModel(trainData,validationData,
                                impurityParm, maxDepthParm, maxBinsParm):
             startTime = time()
             model = DecisionTree.trainClassifier(trainData,
                     numClasses=2, categoricalFeaturesInfo={}, impurity=impurityParm,
                     maxDepth=maxDepthParm, maxBins=maxBinsParm)
             AUC = evaluateModel(model, validationData)
             duration = time() - startTime
             print   "训练评估：使用参数" + \
                     " impurity="+str(impurityParm) +\
                     " maxDepth="+str(maxDepthParm) + \
                     " maxBins="+str(maxBinsParm) +"\n" +\
                      " ==>所需时间="+str(duration) + \
                      " 结果 AUC = " + str(AUC)
             return (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)
```

图 13-46 编写 trainEvaluateModel 函数

以上程序代码说明如表 13-9 所示。

表 13-9 trainEvaluateModel 程序代码说明

指令	详细说明
from time import time	导入 time 模块，计算训练评估所需的时间
def trainEvaluateModel(trainData, validationData, impurityParm, maxDepthParm, maxBinsParm):	定义 trainEvaluateModel 函数，传入下列参数： • trainData：训练时使用 • validationData：验证数据 • DecisionTree：训练时所需要的参数，即 impurity、maxDepth、maxBins
startTime = time()	记录开始时间
model = DecisionTree.trainClassifier(trainData, numClasses=2, categoricalFeaturesInfo={}, impurity=impurityParm, maxDepth=maxDepthParm, maxBins=maxBinsParm)	使用 DecisionTree.trainClassifier 进行训练，返回 model，传入参数： • trainData 训练数据 • numClasses=2 是二元分类 • categoricalFeaturesInfo 空 dict • Impurity 参数 • maxDepth 决策树最大深度参数 • maxBins 决策树节点最大分支数目参数
AUC = evaluateModel (model, validationData)	使用 evaluateModel 进行训练评估模型，传入参数： model 模型与 validationData 验证数据，返回 AUC
duration = time() - startTime	计算所需的时间
print " 训练评估：使用参数 " …	显示运行结果，输入的参数与运行后的 AUC
return (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)	返回 AUC、运行所需的时间，训练参数，训练完成后的模型

步骤 02 运行 trainEvaluateModel

可以使用下列指令，运行 trainEvaluateModel，传入下列参数（见图 13-47）：

```
In [30]: (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \
             trainEvaluateModel(trainData, validationData, "entropy", 5, 5)

训练评估：使用参数 impurity=entropy maxDepth=5 maxBins=5
  ==>所需时间=3.09240007401 结果AUC = 0.656874990538
```

图 13-47　运行 trainEvaluateModel

以上运行结果，显示训练评估模型，所使用的参数以及运行所需的时间，最后为 AUC 评估结果。

13.10.2　评估 impurity 参数

步骤 01 运行 trainEvaluateModel 评估 impurity 参数

我们要评估 impurity 参数的程序代码如图 13-48 所示。其中，for 语句会重复运行 trainEvaluateModel，传入参数 impurityList 有 2 个不同值 ["gini", "entropy"]，固定 maxDepthList=[10]、maxBinsList= [10]，最后把运行结果存入 metrics。

```
In [32]: impurityList=["gini", "entropy"]
         maxDepthList  =[10]
         maxBinsList=[10 ]

         metrics = [trainEvaluateModel(trainData, validationData,
                            impurity,maxDepth,  maxBins  )
                    for impurity in impurityList
                    for maxDepth in maxDepthList
                    for maxBins in maxBinsList ]

训练评估：使用参数 impurity=gini maxDepth=10 maxBins=10
  ==>所需时间=3.41780996323 结果AUC = 0.615456238361
训练评估：使用参数 impurity=entropy maxDepth=10 maxBins=10
  ==>所需时间=2.99922299385 结果AUC = 0.661920334287
```

图 13-48　运行 trainEvaluateModel 评估 impurity 参数

trainEvaluateModel 会执行两次，我们可以看到 impurity =["entropy"]，maxdepth=[10]、maxBins= [10] 具有较高的 AUC。后续将以图表来显示，那样比较容易看出它们之间的关系。

步骤 02 查看 metrics

上一步骤的运行结果存放到了 metrics 中。metrics 的内容如图 13-49 所示。

```
In [33]: metrics
Out[33]: [(0.6713135530043239,
          3.9740288257598877,
          'gini',
          10,
          10,
          DecisionTreeModel classifier of depth 10 with 677 nodes),
         (0.6602959986816601,
          2.8627259731292725,
          'entropy',
          10,
          10,
          DecisionTreeModel classifier of depth 10 with 589 nodes)]
```

第 1 项是参数 impurity = ["gini"]，maxdepth = [10]、maxBins = [10] 的运行结果

第 2 项是参数 impurity = ["entropy"]，maxdepth=[10]、maxBins= [10] 的运行结果

图 13-49 查看 metrics

13.10.3 训练评估的结果以图表显示

后续要将训练评估的结果以图表显示，所以我们要将 metrics 转换为 Pandas DataFrame，并用 Pandas DataFrame 绘图。

步骤01 将 metrics 转换为 Pandas DataFrame

在 IPython Notebook 输入如图 13-50 所示的命令，将 metrics 转换为 Pandas DataFrame。

```
In [34]: import pandas as pd
         IndexList=impurityList
         df = pd.DataFrame(metrics,index=IndexList,
                 columns=['AUC', 'duration','impurity', 'maxDepth', 'maxBins','model'])
         df
```

Out[32]:

	AUC	duration	impurity	maxDepth	maxBins	model
gini	0.651452	3.989273	gini	10	10	DecisionTreeModel classifier of depth 10 with ...
entropy	0.645243	3.304525	entropy	10	10	DecisionTreeModel classifier of depth 10 with ...

DataFrame 的索引，也就是 impurityList

图 13-50 将 metrics 转换为 Pandas DataFrame

以上运行结果显示 Pandas DataFrame。程序代码说明如表 13-10 所示。

表 13-10 程序代码说明

参数	说明
import pandas as pd	导入 pandas 模块
IndexList=impurityList	设置 IndexList 为 impurityList

（续表）

参数	说明
df = pd.DataFrame(metrics, index=IndexList, columns=['AUC','duration' ,'impurity','maxDepth', 'maxBins','model'])	使用 pd.DataFrame 方法转换为 pandas Data Frame，并输入下列参数： • metrics 是要转换的 List • index 设置 pandas dataframe 的索引 • columns 设置 pandas dataframe 的字段
df	查看 pandas data frame

步骤 02 编写 showchart() 函数

之前建立的 pandas Data Frame 可以使用 Matplotlib 绘图，我们将编写 showchart() 函数来显示图表。

```
In [42]: import matplotlib.pyplot as plt
         def showchart(df,evalparm ,barData,lineData,yMin,yMax):
             ax = df[barData].plot(kind='bar', title =evalparm,
                                   figsize=(10,6),legend=True, fontsize=12)
             ax.set_xlabel(evalparm,fontsize=12)
             ax.set_ylim([yMin,yMax])
             ax.set_ylabel(barData,fontsize=12)
             ax2 = ax.twinx()
             ax2.plot(df[lineData ].values, linestyle='-', marker='o',
                      linewidth=2.0,color='r')
             plt.show()
```

上列程序代码说明如下：

➢ **导入模块并定义函数**

指令	说明
import matplotlib.pyplot as plt	导入 matplotlib 模块，并且设置 plt 为别名，后续可用 plt 来引用 matplotlib 模块
def showchart(df, evalparm, barData, lineData, yMin, yMax):	定义 showchart 函数，传入参数： • df 就是之前 metrics 产生的 dataframe • evalparm 是此次评估的参数 • barData 绘出 barchart 数据 • lineData 绘出 linechart 数据，在此是 duration • yMin,yMax 是 *y* 轴的打印区域

➢ **绘制直方图**

指令	说明
ax =df[barData].plot(kind='bar', title=evalparm, figsize=(10,6), legend=True, fontsize=12)	以 dataframe 的 barData 字段进行绘图 图形是 bar chart 设置显示在 *x* 轴 evalparm 参数 设置图形的宽与高 设置显示图标 设置字号
ax.set_xlabel(evalparm, fontsize=12)	设置图形 *x* 轴是 evalparm 参数，并设置字号
ax.set_ylabel(barData,fontsize=12)	设置图形 *y* 轴是 barData 参数
ax.set_ylim([yMin,yMax])	设置 *y* 轴的打印区域

➢ **绘制折线图**

指令	说明
ax2 = ax.twinx()	建立另外一个图形 ax2
ax2.plot(df[lineData].values, linestyle='-', marker='o', linewidth=2.0, color='r')	以 ax2 绘图 以 dataframe 的 duration(运行时间) 字段进行绘图 Line chart 样式 Line chart 数值点是圆形 Line 的宽度 Line 的颜色
plt.show()	开始绘图

步骤03 运行 showchart() 函数

接下来，我们可以运行 showchart() 函数，传入参数：

- 传入 df 参数就是之前 metrics 产生的 dataframe。
- evalparm 当前评估的参数：传入 impurity。
- barData 绘出 barchart 数据：传入 AUC。
- lineData 绘出 linechart 数据：传入 duration 运行时间。
- yMin, yMax 是进行绘图的 *y* 轴范围：传入 0.5, 0.7。

```
In [35]: showchart(df,'impurity','AUC','duration',0.5,0.7 )
```

运行后会打开如图 13-51 所示的绘图窗口。

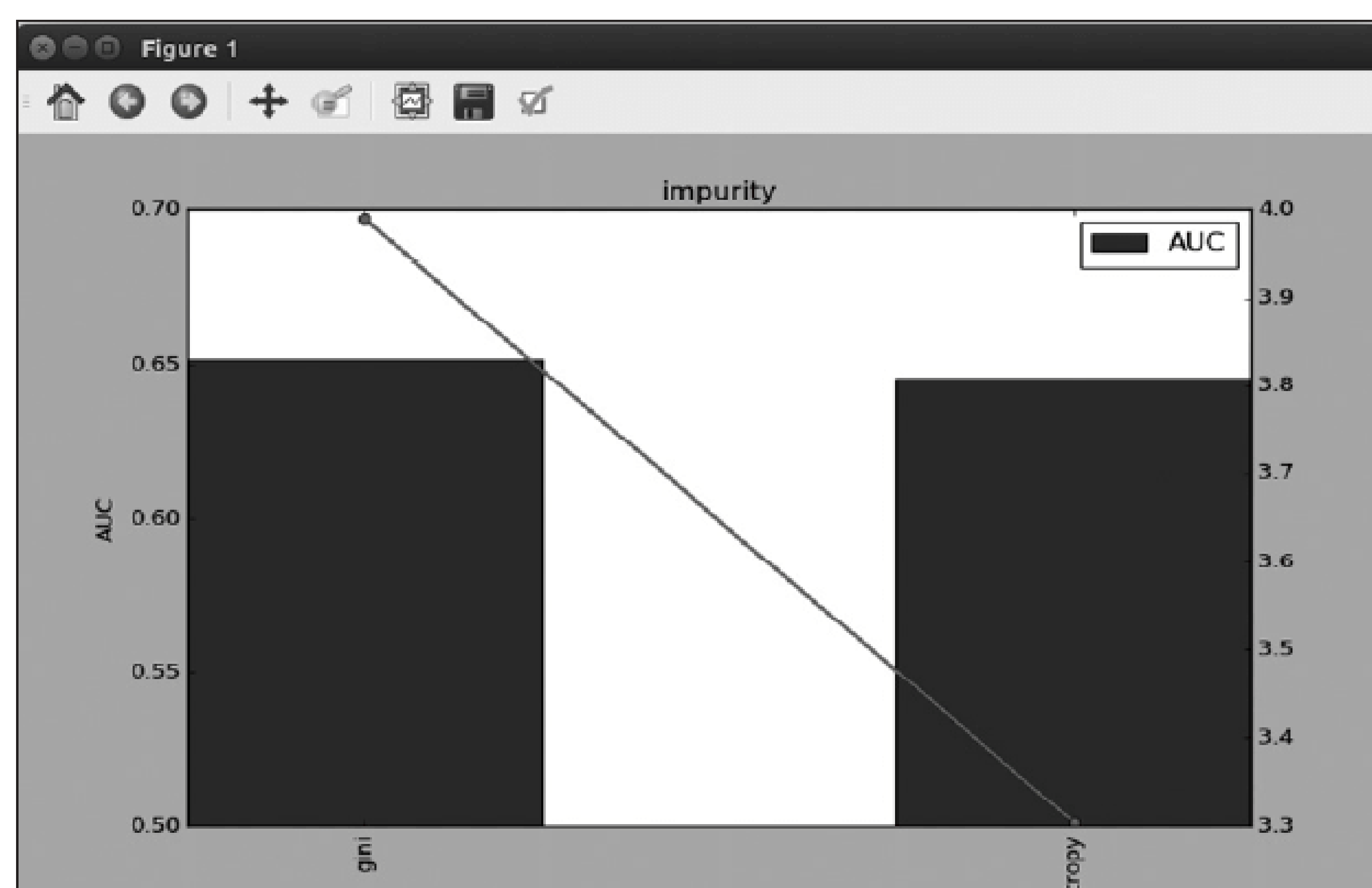


图 13-51　打开绘图窗口

直方图代表 AUC、折线图代表运行时间。从图 13-51 可以看到 entropy 准确度比 gini 好一些，但是并没有很大的差别，而且运行时间 entropy 所需的时间比 gini 少，所以针对这个数据集，选择 entropy 参数是不错的选择。

13.10.4　编写 evalParameter

之前步骤我们评估 'impurity' 参数，并且绘出参数值对准确率的影响以及训练所需的时间。后续我们还需要评估 'maxDepth' 与 'maxBins' 参数，所以我们编写 evalParameter 函数（见图 13-52），可以用来评估不同参数。

```
In [ ]: #定义evalParameter函数
        def evalParameter(trainData, validationData, evalparm,
                    impurityList, maxDepthList, maxBinsList):
          #训练评估参数
          metrics = [trainEvaluateModel(trainData, validationData,
                         impurity,maxDepth,  maxBins  )
                     for impurity in impurityList
                     for maxDepth in maxDepthList
                     for maxBins in maxBinsList ]
          #设置当前评估的参数'
          if evalparm=="impurity":
            IndexList=impurityList[:]
          elif evalparm=="maxDepth":
            IndexList=maxDepthList[:]
          elif evalparm=="maxBins":
            IndexList=maxBinsList[:]
          #转换为Pandas DataFrame
          df = pd.DataFrame(metrics,index=IndexList,
              columns=['AUC', 'duration','impurity', 'maxDepth', 'maxBins','model'])
          #显示图形
          showchart(df,'impurity','AUC','duration',0.5,0.7 )
```

图 13-52　编写 evalParameter 函数

13.10.5　使用 evalParameter 评估 maxDepth 参数

当我们要评估 maxDepth 参数时，固定 impurityList =["gini"]、maxBinsList=[10]，但是

maxDepthList=[3, 5, 10, 15, 20, 25]，传入 maxdepthArray 值来评估哪一个参数具有比较好的准确率（也就是 AUC 值比较高）。

```
In [*]: evalParameter(trainData, validationData,"maxDepth",
                      impurityList=["gini"],
                      maxDepthList=[3, 5, 10, 15, 20, 25],
                      maxBinsList=[10])
```

运行后结果如图 13-53 所示，并且会打开如图 13-54 所示的绘图窗口。

```
训练评估：使用参数 impurity=gini maxDepth=3 maxBins=10
 ==>所需时间=2.38977503777 结果AUC = 0.591330941242
训练评估：使用参数 impurity=gini maxDepth=5 maxBins=10
 ==>所需时间=2.30682897568 结果AUC = 0.642950144585
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=10
 ==>所需时间=2.52588891983 结果AUC = 0.615456238361
训练评估：使用参数 impurity=gini maxDepth=15 maxBins=10
 ==>所需时间=2.94089603424 结果AUC = 0.61464247324
训练评估：使用参数 impurity=gini maxDepth=20 maxBins=10
 ==>所需时间=3.060505867 结果AUC = 0.598132503671
训练评估：使用参数 impurity=gini maxDepth=25 maxBins=10
 ==>所需时间=3.40477705002 结果AUC = 0.595380842077
```

图 13-53　评估 maxDepth 参数

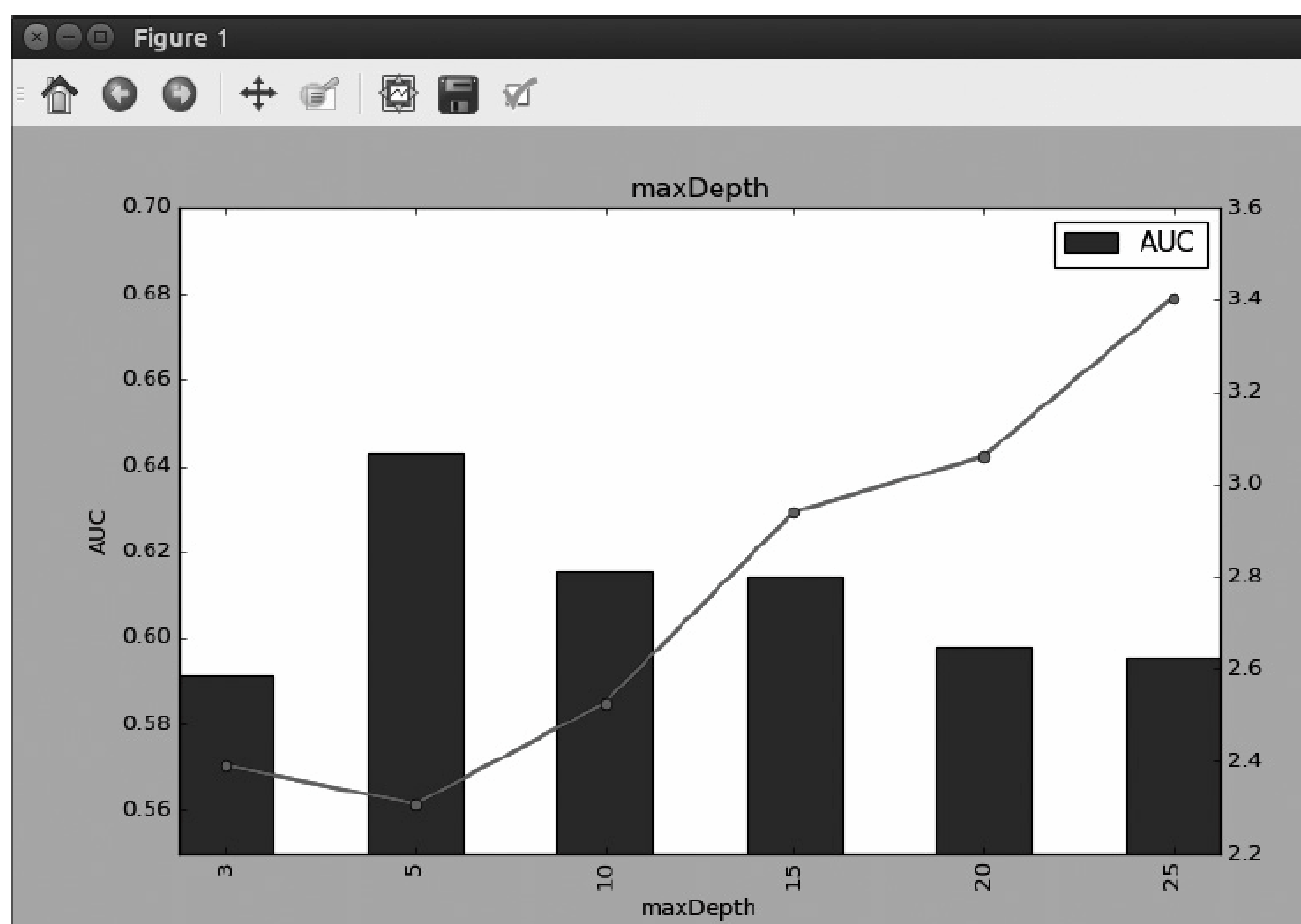


图 13-54　打开绘图窗口

从图 13-54 可以看到，maxDepth=5，AUC 最高，maxDepth 越大，所需时间越多，所以 maxDepth=5 可能是不错的选择。

13.10.6　使用 evalParameter 评估 maxBins 参数

当我们要评估 maxBins 时，固定 impurity=["gini"]、maxDepth= [10]，但是 maxBins 输入参数有 6 个不同值 [3, 5, 10, 50, 100, 200]，用来评估哪一个参数具有比较好的 准确率——也



就是 AUC 值比较高。

```
In [*]: evalParameter(trainData, validationData,"maxBins",
                  impurityList=["gini"],
                  maxDepthList =[10],
                  maxBinsList=[3, 5, 10, 50, 100, 200 ])
```

运行后结果如图 13-55 所示，同时会打开如图 13-56 所示的窗口。

```
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=3
 ==>所需时间=4.30378603935 结果AUC = 0.627295574632
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=5
 ==>所需时间=2.84377002716 结果AUC = 0.65481219058
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=10
 ==>所需时间=3.02329683304 结果AUC = 0.615456238361
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=50
 ==>所需时间=3.31911301613 结果AUC = 0.644952385278
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=100
 ==>所需时间=3.19374299049 结果AUC = 0.636863938472
训练评估：使用参数 impurity=gini maxDepth=10 maxBins=200
 ==>所需时间=3.71806001663 结果AUC = 0.641386958562
```

图 13-55　评估 maxBins 参数

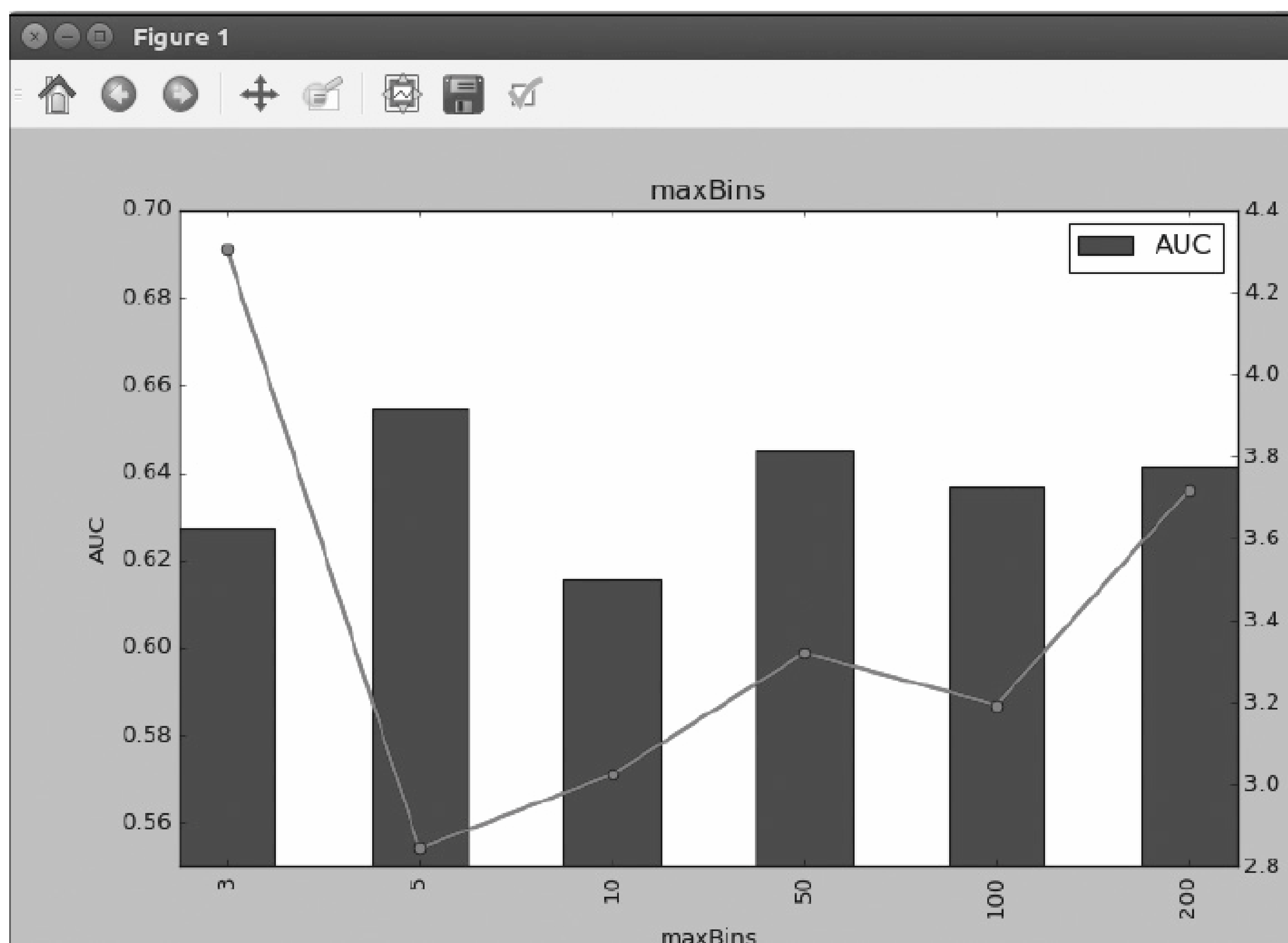


图 13-56　打开绘图窗口

从图 13-56 可以看到，maxBins=5 时 AUC 最高，maxBins 越大，所需时间越多，所以 maxBins=5 可能是不错的选择。

13.11 如何找出准确率最高的参数组合

之前的章节我们对各个参数进行评估，了解每一个参数值对 AUC 与运行时间的关系。接下来，我们将所有参数训练评估找出最好的参数组合。

执行的步骤如下：

（1）执行训练 DecisionTree.trainClassifier 传入 trainData、所有参数组合。
（2）建立模型后，以 validationData 评估模型的 AUC 准确率。
（3）训练与评估模型重复执行多次，产生多项的 AUC 与运行时间，并存储在 AUC metrics RDD 中。
（4）全部执行完成后，AUC metrics 取出 AUC 最大的参数组合，就是最佳模型 bestModel。

程序流程示意图如图 13-57 所示。

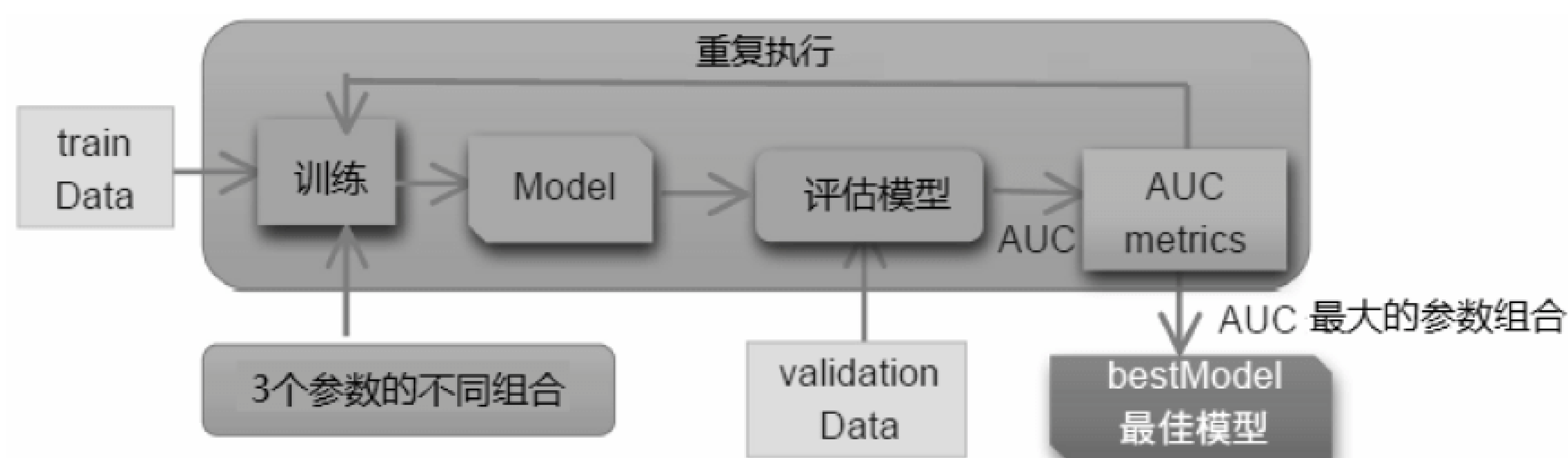


图 13-57　程序流程示意图

步骤 01 编写 evalAllParameter 函数

我们将编写 evalAllParameter 函数，所有参数训练评估找出最好的参数组合，如图 13-58 所示。

```
In [58]:  #定义evalAllParameter函数
          def evalAllParameter(trainData, validationData,
                     impurityList, maxDepthList, maxBinsList):
              #for循环训练评估所有参数组合
              metrics = [trainEvaluateModel(trainData, validationData,
                              impurity,maxDepth,  maxBins  )
                         for impurity in impurityList
                         for maxDepth in maxDepthList
                         for  maxBins in maxBinsList ]
              #找出AUC最大的参数组合
              Smetrics = sorted(metrics, key=lambda k: k[0], reverse=True)
              bestParameter=Smetrics[0]
              #显示调校后最佳参数组合
              print(" 调校后最佳参数  : impurity:" + str(bestParameter[2]) +
                                   ",maxDepth:" + str(bestParameter[3]) +
                                  ",maxBins:" + str(bestParameter[4])  +
                                   "\n,  结果AUC  = " + str(bestParameter[0]))
              #返回最佳模型
              return bestParameter[5]
```

图 13-58　编写 evalAllParameter 函数

➢ **调用 evaluateAllParameter 函数的方式（见图 13-59）**

```
In [59]:  print("----所有参数训练评估找出最好的参数组合---------")
          bestModel=evalAllParameter(trainData, validationData,
                         ["gini", "entropy"],
                         [3, 5, 10, 15, 20, 25],
                         [3, 5, 10, 50, 100, 200 ])
```

图 13-59　调用 evaluateAllParameter 函数

以上程序代码说明如下：

#定义evalAllParameter函数

- 传入参数：trainData 训练数据、validationData 验证数据、impurityArray 参数、maxDepthArray 参数、maxBinsArray 参数。
- 返回 DecisionTreeModel 模型。

#for循环训练评估所有参数组合

- 3 个 for 循环读取 List，即 impurityList、maxDepthArray、maxBinsArray，3 个 List 排列组合，共有 2*6*6=72 种排列组合，会重复执行 trainEvaluateModel 共 72 次。
- 产生 72 项数据的 List，将(AUC, duration, impurityParm, maxDepthParm, maxBinsParm, model) 所组成的 List 存入 metrics。

#找出AUC最大的参数组合

之前已产生 72 种排列组合的 metrics，所以我们希望在 72 组参数组合中找出 AUC 中最大的参数组合。所以我们使用下列指令先从大到小排序，然后取出第 1 组数据，就是 AUC 最大的 1 组数据。

- Smetrics = sorted(metrics, key=lambda k: k[0], reverse=True)，进行从大到小排序。
- bestParameter=Smetrics[0]，获取第 1 组数据，就是 AUC 最大，也就是最佳参数组合。

#返回最佳模型

从 0 算起第 6 个字段是训练完成的最佳模型，所以返回 bestParameter[5]。

步骤 02 执行 evalAllParameter 函数

执行 evalAllParameter 函数，如图 13-60 所示。

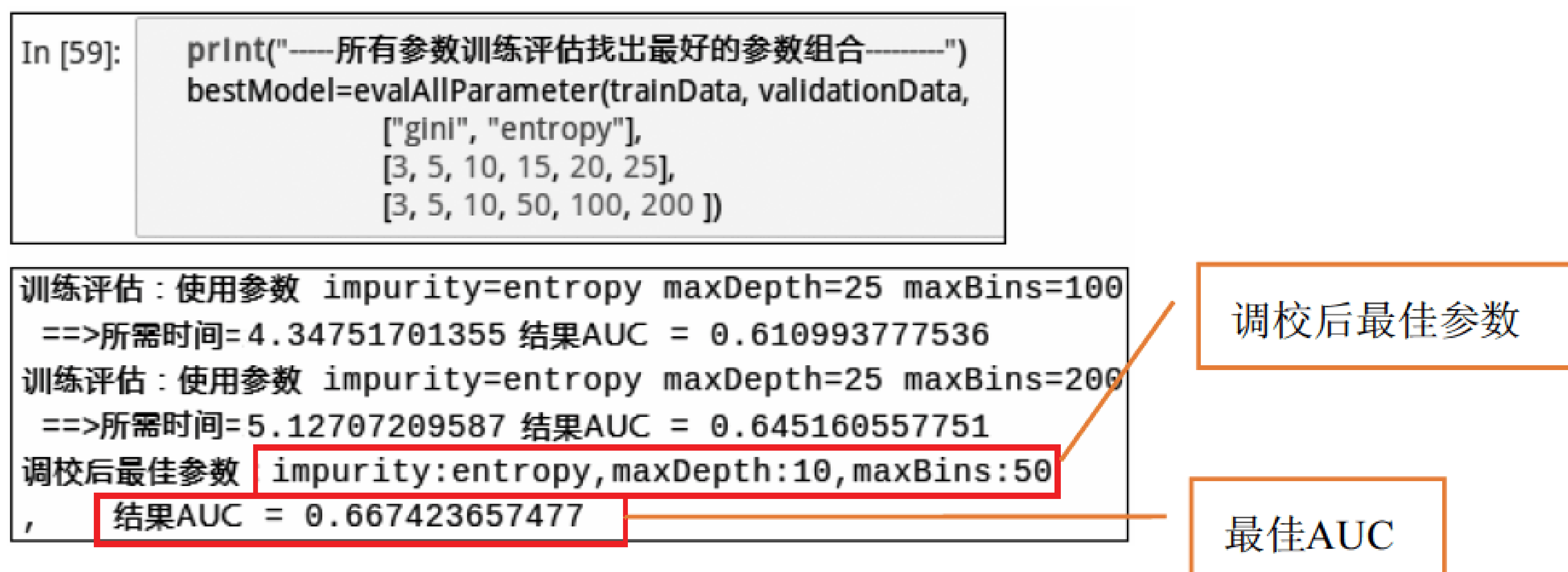


图 13-60 执行 evalAllParameter 函数

以上执行结果会显示 72 组数据，我们仅列出最后几组数据，最后会显示最佳的参数组合。

13.12 如何确认是否过度训练

所谓过度训练（Overfitting），是指机器学习所学到的模型过度贴近 trainData，从而导致误差变得很大。为了确认没有 Overfitting（过度训练）问题：

- 首先，在训练评估阶段时使用 validationData 评估模型。
- 然后，在测试阶段时使用另外一组数据 testData 测试数据后再测试模型。

如果训练评估阶段时 AUC 很高，但是测试阶段 AUC 很低，就代表可能有 overfitting 的问题。以下程序代码使用 testData 进行评估：

```
In [33]: AUC=evaluateModel(model, testData)
         print "AUC="+str(AUC)

AUC=0.667698395358
```

以上测试结果显示 AUC 大约为 0.66，与训练阶段差异不大，代表没有 overfitting 的问题。

13.13 编写 RunDecisionTreeBinary.py 程序

添加 RunDecisionTreeBinary.py（见图 13-61）

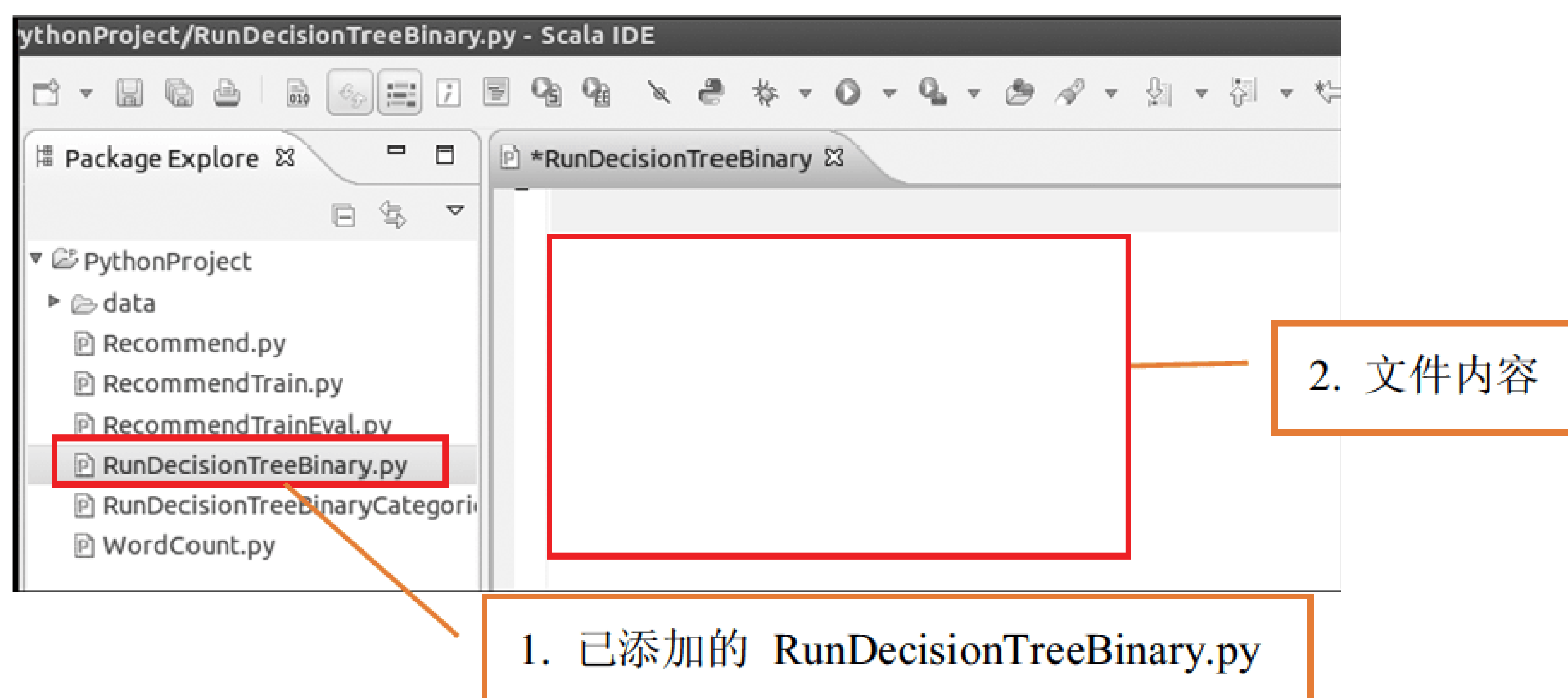


图 13-61 添加 RunDecisionTreeBinary.py

13.14 开始输入 RunDecisionTreeBinary.py 程序

请参考本书范例 RunDecisionTreeBinary.py，其中大部分的程序代码在之前的章节都已经使用 IPython Notebook 演练过了，在此仅说明 main 主程序。

输入 main 主程序

输入 main 主程序如下：

```
if  name == " main ":
  print("RunDecisionTreeBinary") sc=CreateSparkContext()
  print("========== 数据准备阶段 ===============")
  (trainData, validationData, testData, categoriesMap) =PrepareData(sc)
  trainData.persist(); validationData.persist(); testData.persist()
  print("========== 训练评估阶段 ===============")
  (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \
        trainEvaluateModel(trainData, validationData, "entropy", 5, 5)
  if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
      parametersEval(trainData, validationData)
  elif  (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
      print("----- 所有参数训练评估找出最好的参数组合 ---------")
      model=evalAllParameter(trainData, validationData,
                       ["gini", "entropy"],
                       [3, 5, 10, 15, 20, 25],
                       [3, 5, 10, 50, 100, 200 ])
  print("========== 测试阶段 ===============")
  auc = evaluateModel(model, testData)
  print(" 使用 test Data 测试最佳模型 , 结果 AUC:" + str(auc))
  print("========== 预测数据 ===============")
  PredictData(sc, model, categoriesMap)
  print model.toDebugString()
```

main function 是程序运行的起点。main function 主要分为 4 个步骤，如表 13-11 所示。

表 13-11　main function 的步骤说明

步骤	说明
数据准备阶段	(trainData, validationData, testData,categoriesMap) =PrepareData(sc) （1）程序会读取文本文件，经过数据转换产生 LabeledPoint RDD 数据 （2）将数据以随机方式分为 3 个部分（trainData、validationData、testData）并返回数据，作为下一阶段训练评估使用 （3）我们还会返回 categoriesMap（网页分类的字典或对照表）作为后续预测时使用，后续章节进行说明 （4）为了增加运行效率，我们使用 persist 命令暂存在内存中： trainData.persist(); validationData.persist(); testData.persist()

（续表）

步骤	说明
训练评估阶段	➢ 如果不输入参数，执行训练评估 (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \ trainEvaluateModel(trainData, validationData, "entropy", 5, 5) ➢ 如果输入参数“-e”，执行参数评估，以图表显示参数与准确率及训练时间的关系 parametersEval(trainData, validationData) ➢ 如果输入参数“-a”，所有参数训练评估找出最好的参数组合 model=evalAllParameter(trainData, validationData, ["gini", "entropy"], [3, 5, 10, 15, 20, 25], [3, 5, 10, 50, 100, 200])
测试阶段	auc = evaluateModel(model, testData) （1）使用另外一组数据 testData 再次测试，以避免 overfitting 的问题 （2）Overfitting（过度训练）是指机器学习所学到的模型过度贴近 trainData，从而导致误差变得更大 （3）如果训练评估阶段时 AUC 很高，但是测试阶段 AUC 很低，就代表可能有 overfitting 的问题 （4）如果测试与训练评估阶段的结果 AUC 差异不大，就代表无 overfitting 的问题
预测阶段	PredictData(sc, model,categoriesMap) 新的数据进行处理后，使用训练完成的模型进行预测

RunDecisionTreeBinary.py 程序架构如图 13-62 所示。

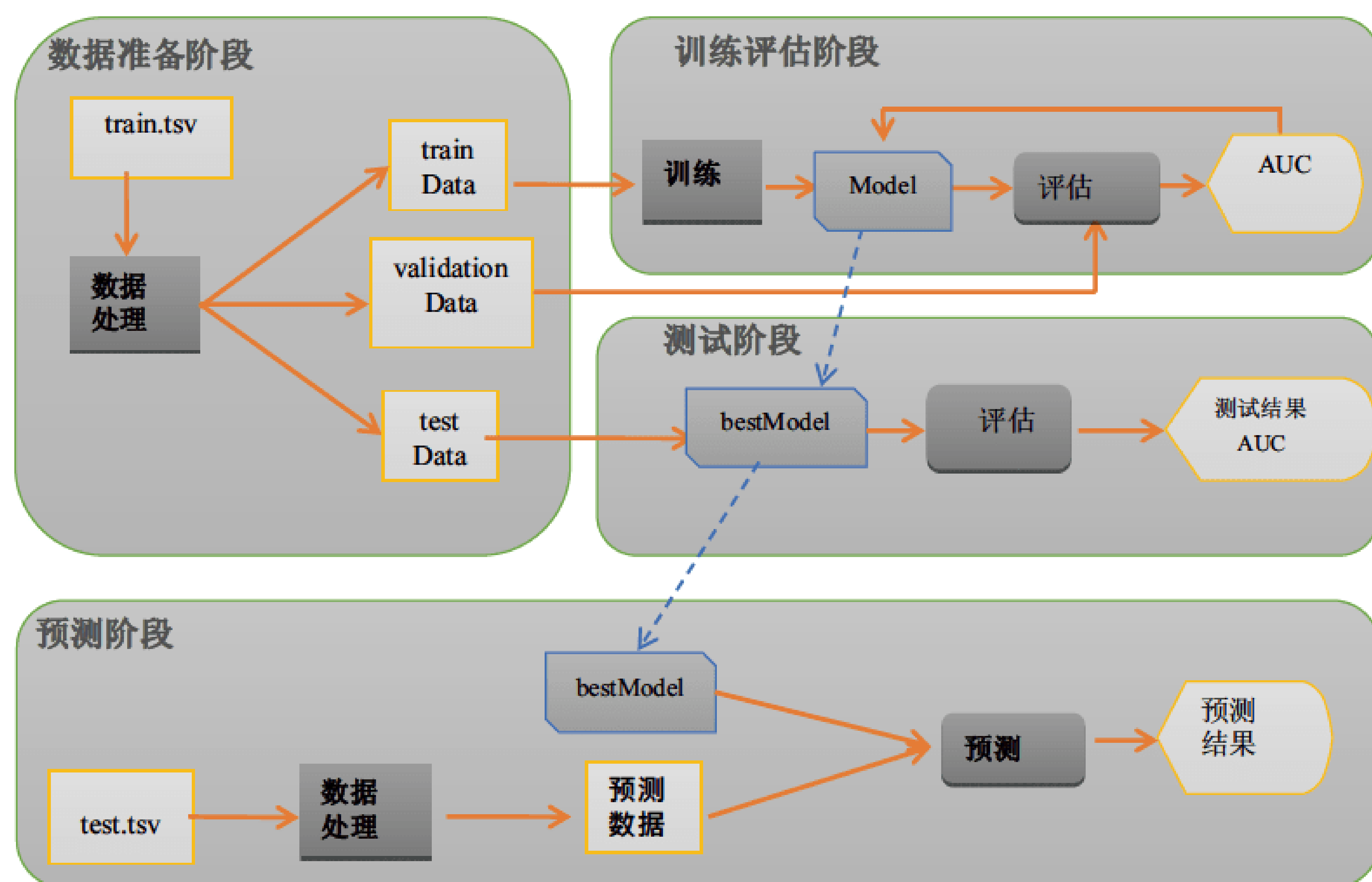


图 13-62　RunDecisionTreeBinary.py 程序架构

13.15　运行 RunDecisionTreeBinary.py

13.15.1　执行参数评估

步骤 01　运行外部工具（见图 13-63）

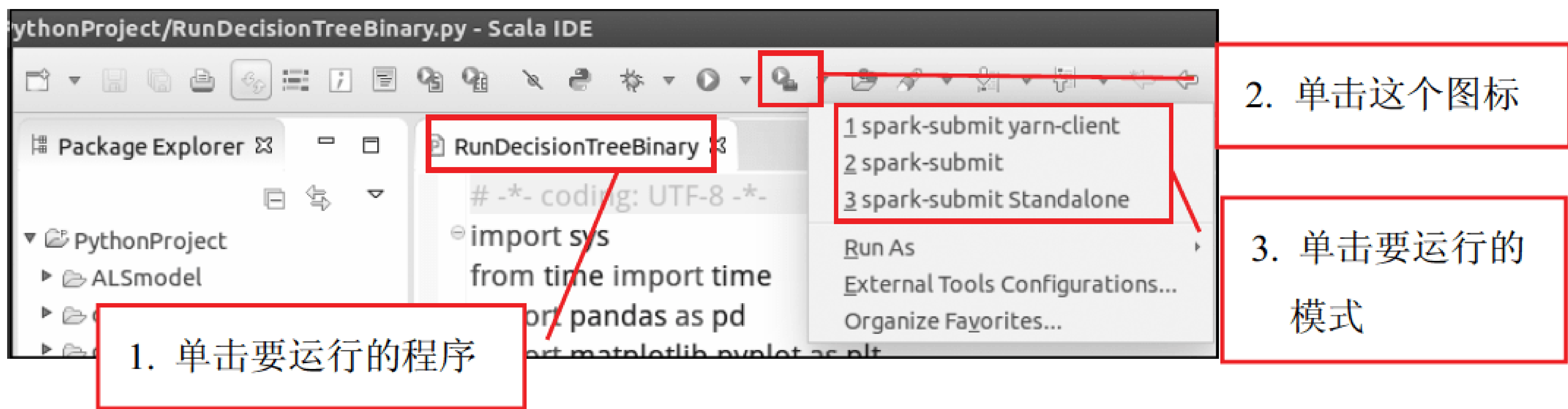


图 13-63　运行外部工具

步骤 02　输入参数 “-e”（见图 13-64）



图 13-64　输入参数

步骤 03　评估 impurity、maxDepth、maxBins 参数

运行后会产生绘图（见图 13-65），可参考 13.10 节的说明。

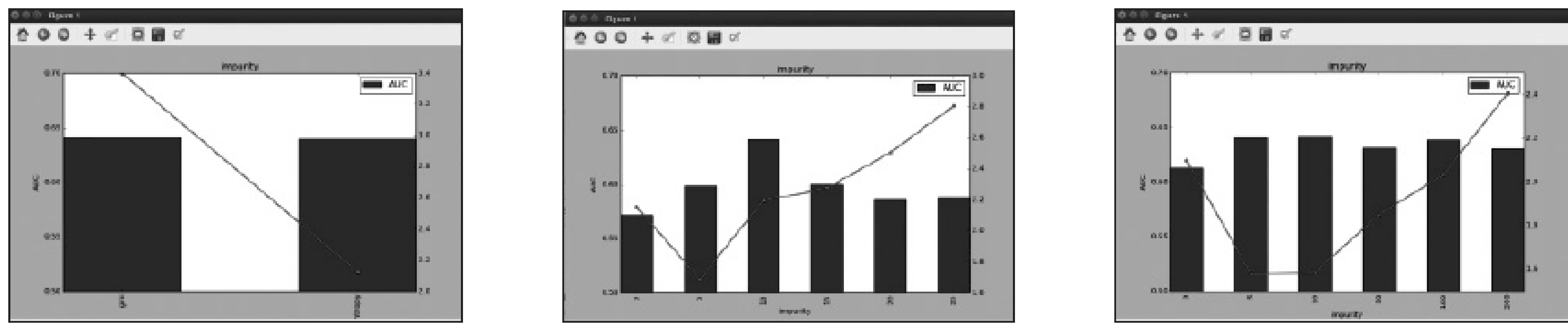


图 13-65　评估 impurity、maxDepth、maxBins 参数

13.15.2　所有参数训练评估找出最好的参数组合

接下来，我们将所有参数训练评估找出最好的参数组合。

步骤01 运行外部工具（见图 13-66）

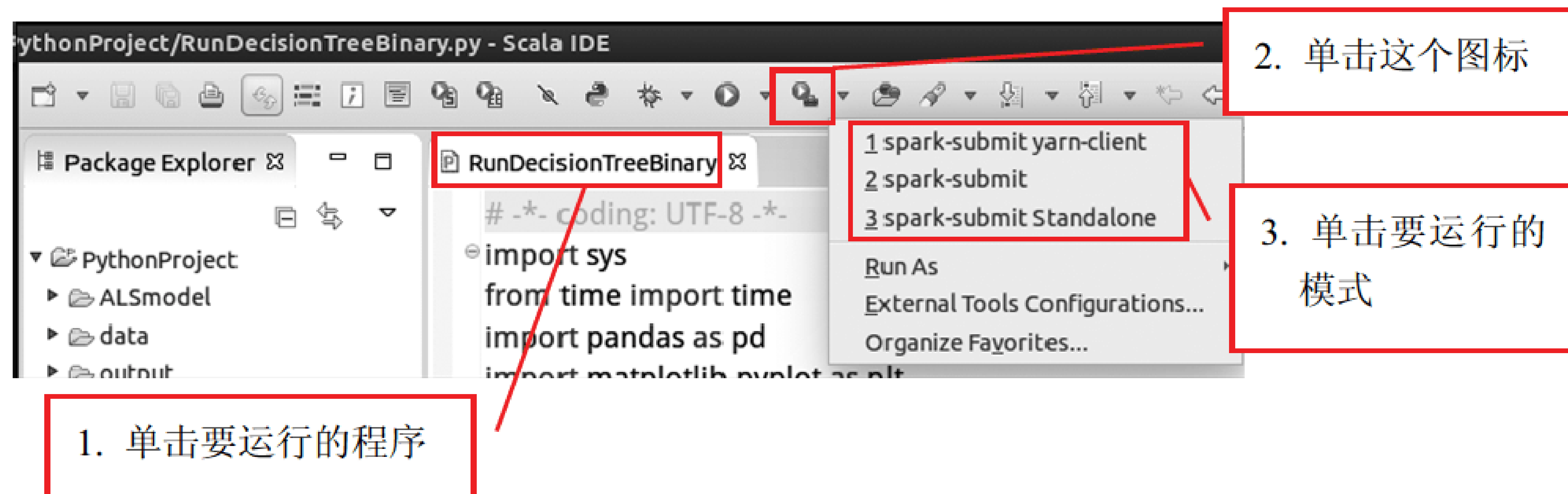


图 13-66 运行外部工具

步骤02 输入参数“-a”（见图 13-67）



图 13-67 输入参数“-a”

步骤03 运行后的屏幕显示界面（见图 13-68）

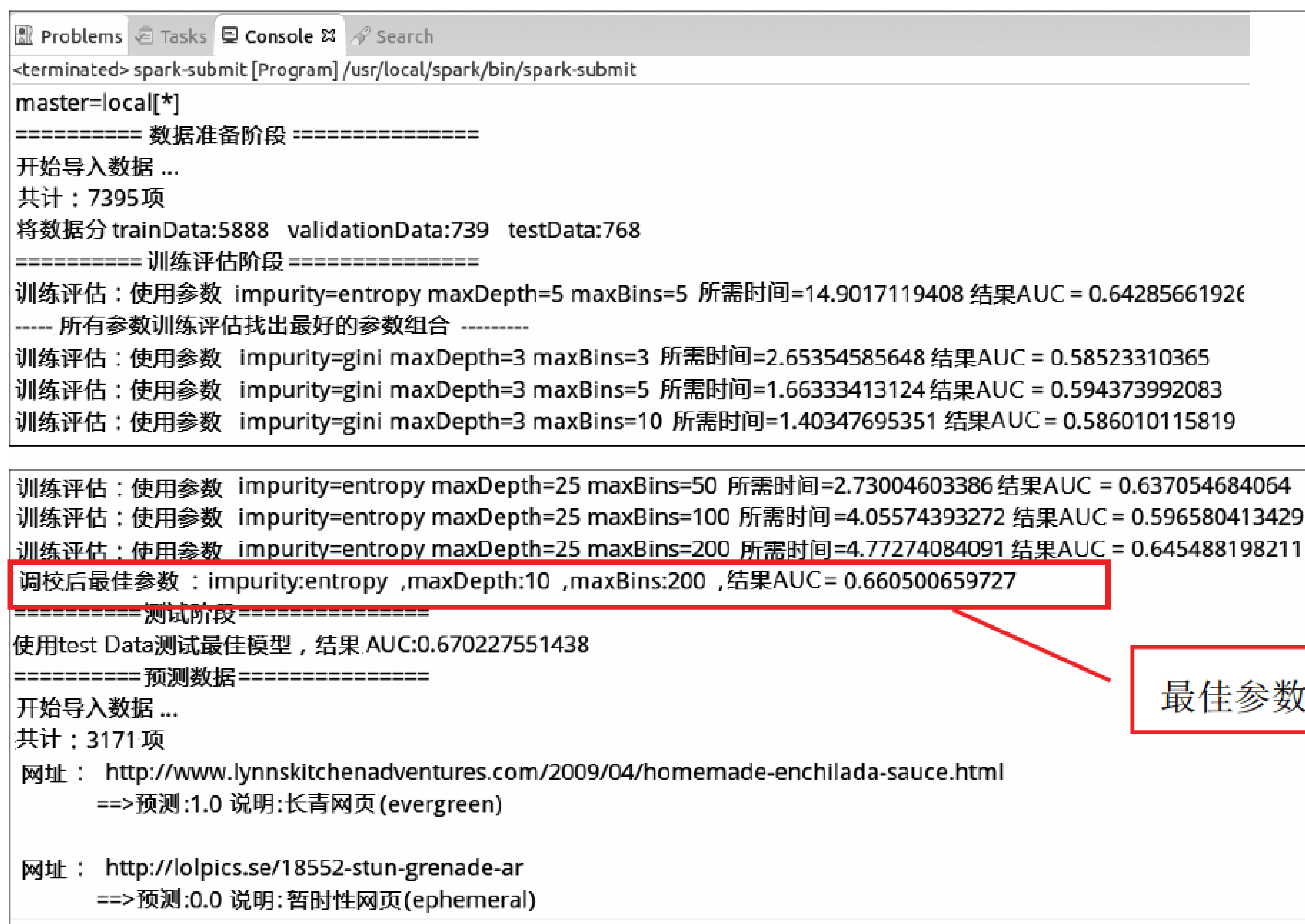


图 13-68 运行后的屏幕显示界面

运行后我们可以看到最佳参数组合是 impurity=entropy,maxDepth=10,maxBins=200，运行结果 AUC 大约是 0.66。

13.15.3　运行 RunDecisionTreeBinary.py 不要输入参数

步骤 01 修改 trainEvaluateModel 程序，改为最佳参数组合（见图 13-69）

因为找出最佳参数组合比较花时间，所以当我们已经找出最佳参数组合时，可以修改 trainEvaluate 程序，改为最佳参数组合。下次执行预测时可以不再执行参数调校，直接使用最佳参数进行预测。

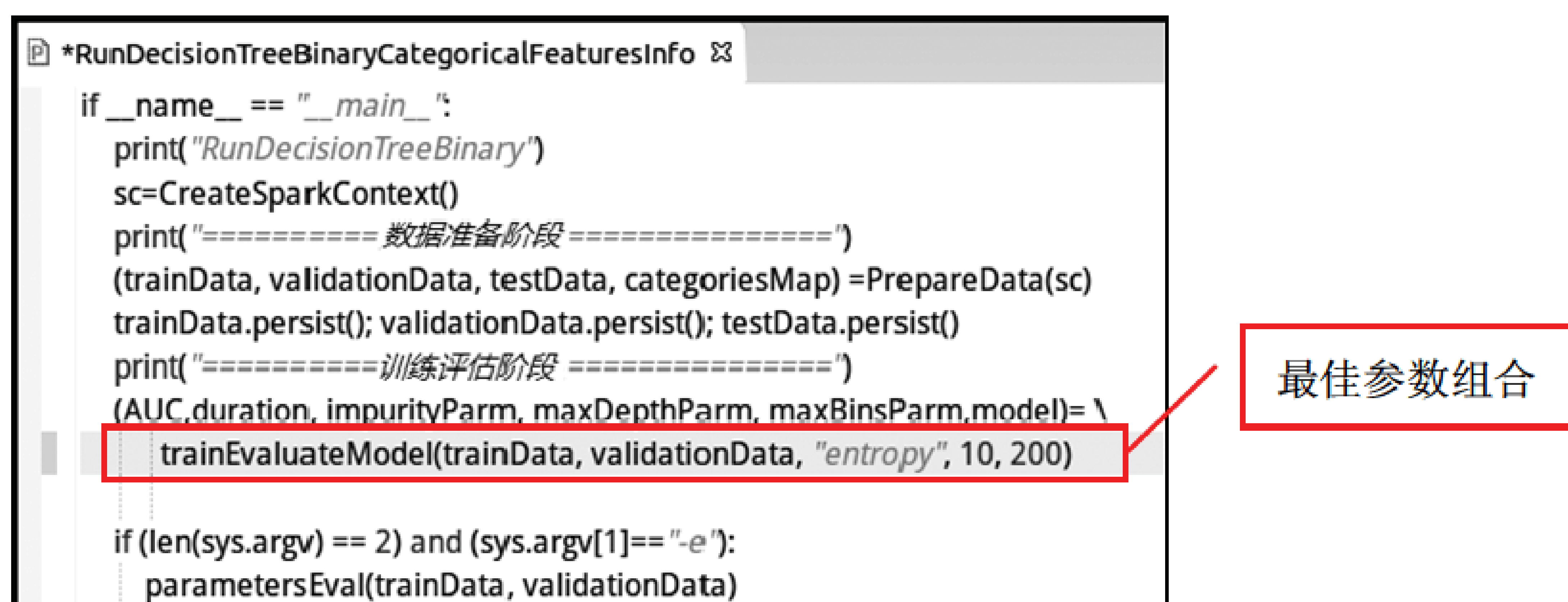


图 13-69　修改为最佳参数组合

步骤 02 运行外部工具（见图 13-70）

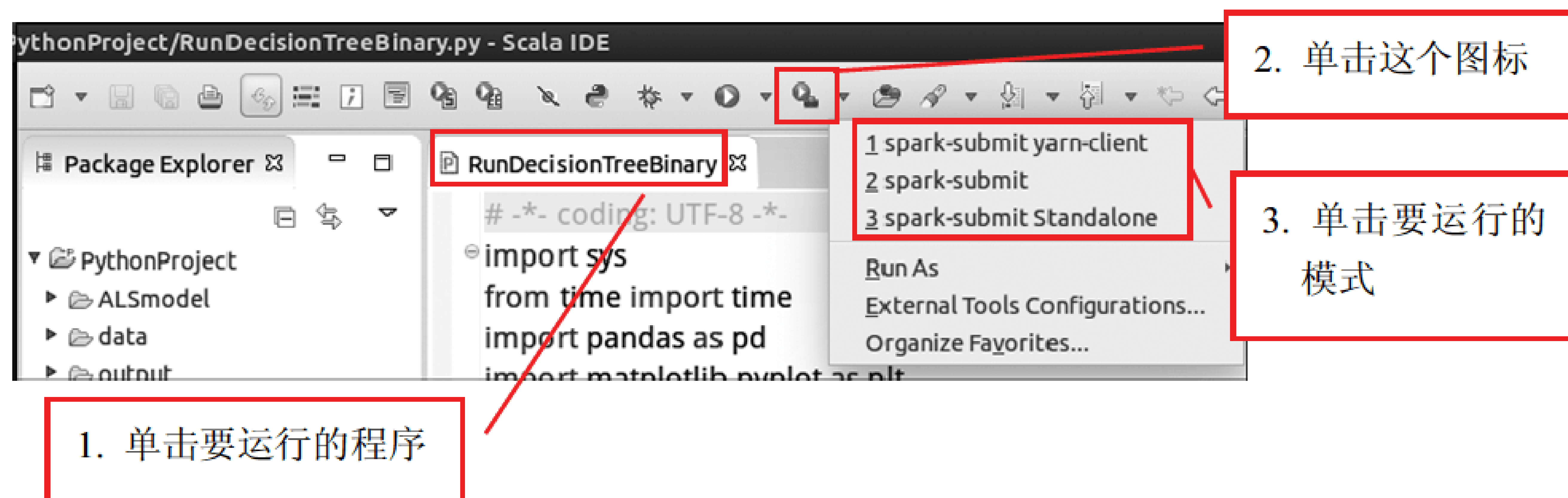


图 13-70　运行外部工具

步骤 03 运行 RunDecisionTreeBinary.py 不要输入参数（见图 13-71）



图 13-71　不输入参数

步骤 04 运行后的屏幕显示界面（见图 13-72）

```
Problems  Tasks  Console  Search
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit
master=local[*]
========== 数据准备阶段 ===============
开始导入数据 ...
共计：7395项
将数据分trainData:5930   validationData:726   testData:739
========== 训练评估阶段 ===============
训练评估：使用参数 impurity=entropy maxDepth=10 maxBins=200 所需时间=11.2442879677 结果AUC = 0.665232240437
==========测试阶段 ===============
使用test Data测试最佳模型，结果AUC:0.636484901788
========== 预测数据 ===============
开始导入数据 ...
共计：3171项
 网址：  http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html
        ==>预测:1.0 说明:长青网页(evergreen)

 网址：  http://lolpics.se/18552-stun-grenade-ar
        ==>预测:0.0 说明:暂时性网页(ephemeral)

 网址：  http://www.xcelerationfitness.com/treadmills.html
        ==>预测:1.0 说明:长青网页(evergreen)
```

1. 数据准备阶段
2. 训练评估阶段
3. 测试阶段
4. 预测数据

图 13-72　运行后的屏幕显示界面

运行结果说明如下：

（1）**数据准备阶段：**显示导入数据项数以及 trainData validationData testData 项数。

（2）**训练评估阶段：**执行最佳参数 impurity=entropy maxDepth=10 maxBins=200，以及评估 结果 AUC（大约 0.66）。

（3）**测试阶段：**使用测试数据再测试模型。AUC 大约为 0.63，与训练阶段差异不大，确认没有 overfitting 的问题。

（4）**预测阶段：**使用 test.tsv 预测是长青网页（evergreen）还是暂时性网页（ephemeral）。预测的结果显示出了网址。我们可以使用浏览器查看这些网址，检查预测的结果与人工判断是否相符。

13.16 查看 DecisionTree 的分类规则

步骤 01 修改 RunDecisionTreeBinary.py 加入查看分类规则

在 main 程序的最下面加入 model.toDebugString()，以查看 DecisionTree 的分类规则（见图 13-73）。



```
(AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \
    trainEvaluateModel(trainData, validationData, "entropy", 5, 5)
if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
    parametersEval(trainData, validationData)
elif   (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
    print("----所有参数交叉评估找出最好的参数组合---------")
    model=evalAllParameter(trainData, validationData,
                    ["gini", "entropy"],
                    [3, 5, 10, 15, 20, 25],
                    [3, 5, 10, 50, 100, 200 ])
print("==========测试阶段===============")
auc = evaluateModel(model, testData)
print("使用test Data测试最佳模型, 结果 AUC :" + str(auc))
print("==========预测数据===============")
PredictData(sc, model, categoriesMap)
print   model.toDebugString()
```

加入 model.toDebugString()

图 13-73　加入查看分类规则

步骤 02　运行外部工具（见图 13-74）

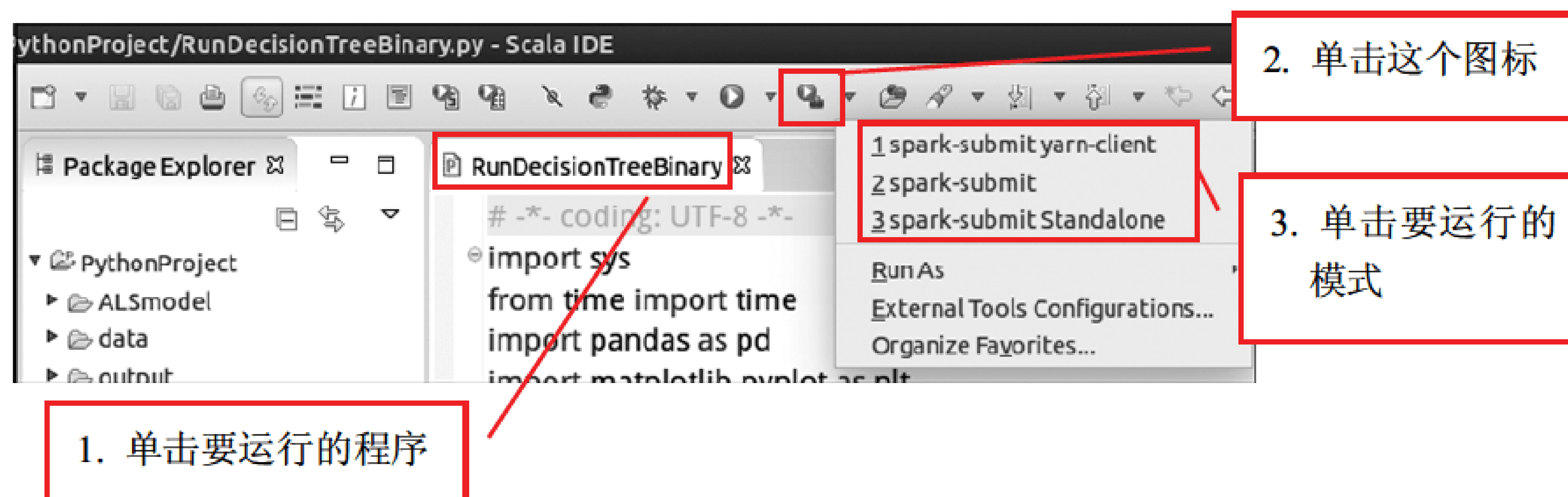


图 13-74　运行外部工具

步骤 03　运行 RunDecisionTreeBinary.py 查看分类规则（见图 13-75）

```
Console
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit

网址： http://www.wired.com/gadgetlab/2011/12/stem-turns-lemons-and-limes-into-juicy-atomizers/
    ==> 预测:0.0 说明：暂时性网页(ephemeral)

网址： http://www.latimes.com/health/boostershots/la-heb-fat-tax-denmark-20111013,0,2603132.story
    ==> 预测:0.0 说明：暂时性网页(ephemeral)

网址： http://www.howlifeworks.com/a/a?AG_ID=1186&cid=7340ci
    ==> 预测:1.0 说明:长青网页(evergreen)

网址： http://romancingthestoveblog.wordpress.com/2010/01/13/sweet-potato-ravioli-with-lemon-sage-brown-butter-sauce/
    ==> 预测:1.0 说明:长青网页(evergreen)

网址： http://www.funniez.net/Funny-Pictures/turn-men-down.html
    ==> 预测:0.0 说明：暂时性网页(ephemeral)

网址： http://youfellasleepwatchingadvd.com/
    ==> 预测:0.0 说明：暂时性网页(ephemeral)

DecisionTreeModel classifier of depth 5 with 61 nodes
 If (feature 31 <= 1203.0)
  If (feature 5 <= 0.0)
   If (feature 13 <= 0.0)
    If (feature 23 <= 0.038331454)
     If (feature 0 <= 0.0)
      Predict: 0.0
     Else (feature 0 > 0.0)
      Predict: 1.0
    Else (feature 23 > 0.038331454)
     If (feature 15 <= 1.912820513)
      Predict: 0.0
     Else (feature 15 > 1.912820513)
```

分类规则

图 13-75　查看分类规则

步骤 04 分类规则说明

运行后产生的分类规则以 If Else 表示。分类规则有很多层，这里仅说明两层，如图 13-76 所示。

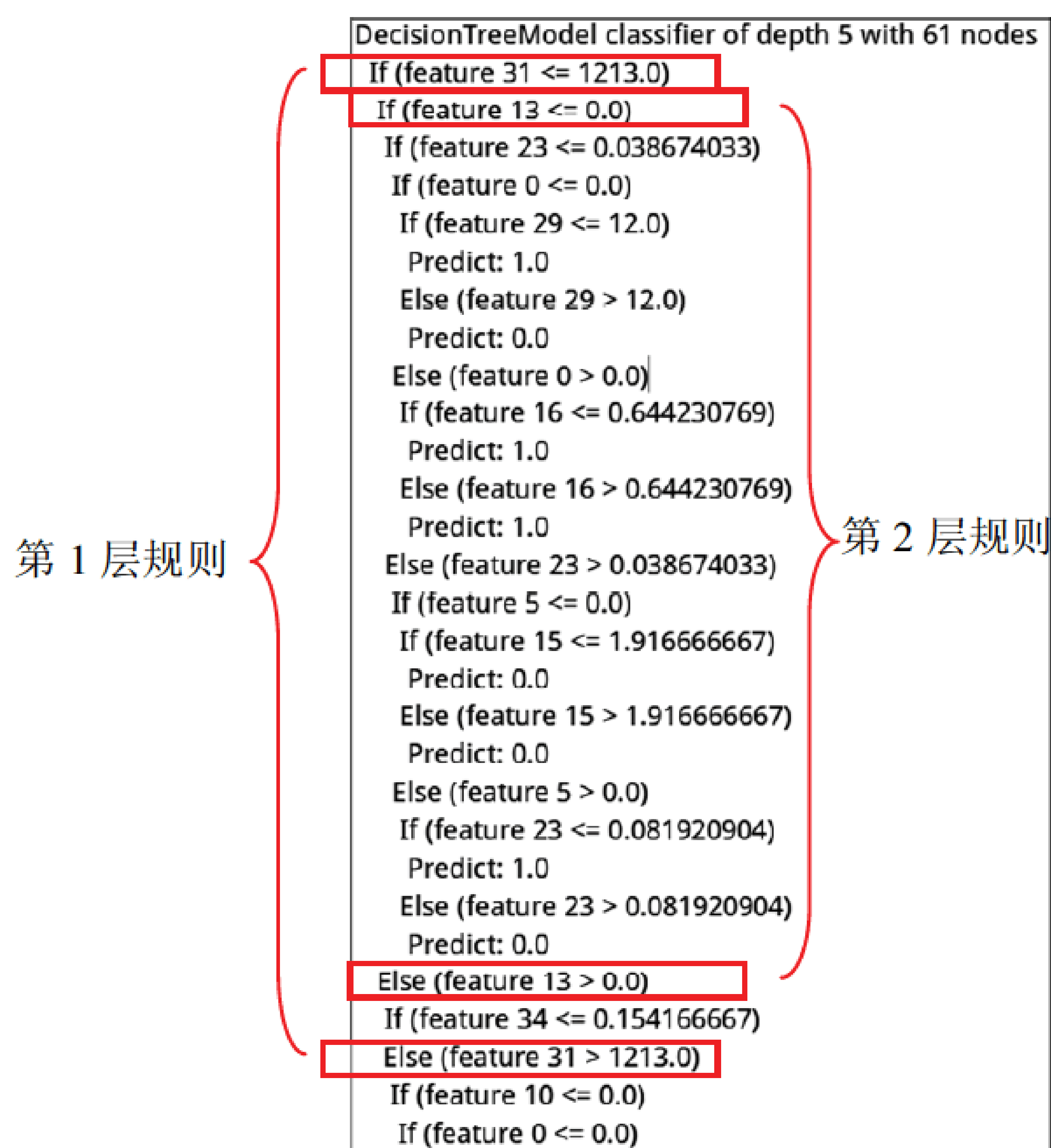


图 13-76　两层分类规则

我们可以画出类似图 13-77 所示的树形图（只画出两层）。

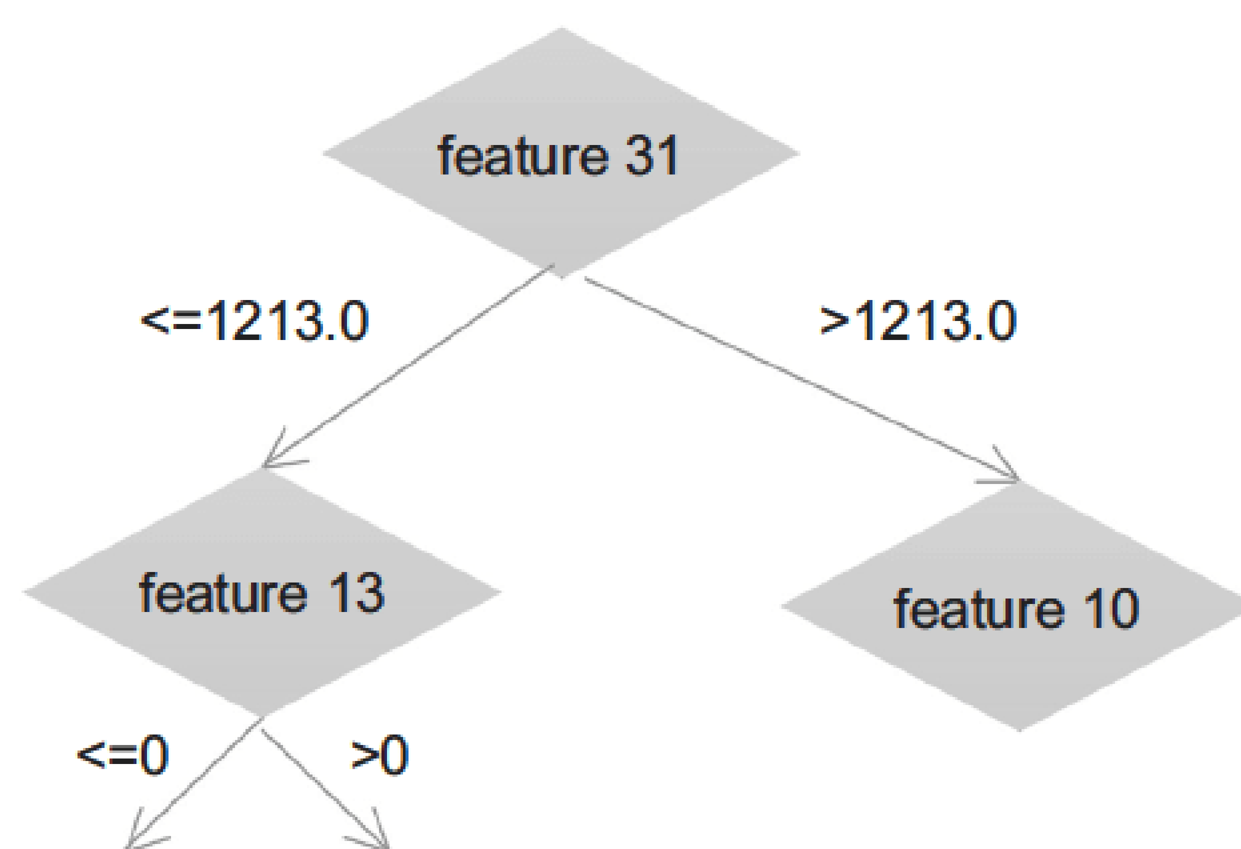


图 13-77　画出树形图

13.17 结论

本章介绍了在 Spark MLlib 支持决策树二元分类的基本原理，完成了数据准备、训练模型、预测网页是暂时性的或是长青的，并调校参数找出最佳参数组合，提高预测准确度。下一章我们将介绍逻辑回归二元分类。

第 14 章 Python Spark MLlib 逻辑回归二元分类

在线性回归中“因变量”是连续变项。例如，年龄与疾病的关系是线性回归，随着年龄增长，得到某疾病的概率也会增加。然而，如果“因变量”不是连续变项，而是二分变项（例如某个年龄是否得到某疾病），就必须使用逻辑回归（Logistic Regression）了。

本章我们将使用第 13 章所介绍的 StumbleUpon 数据集，以逻辑回归二元分类预测网页是暂时性的（ephemeral）还是长青的（evergreen），并训练评估找出最佳参数组合，提高预测准确度。

14.1 逻辑回归分析介绍

先看看简单线性回归（Simple Linear Regression）。

➢ **简单回归分析**

它是假设变量y的数值是自变量x所组成的某种线性函数值再加上一个误差值所得到的数值，如以下的数学公式：

$$y = b_0 + b_1 x$$

例如，年龄与疾病的关系就是线性回归的例子。随着年龄的增长，得某疾病的概率也会增加，如图 14-1 所示。

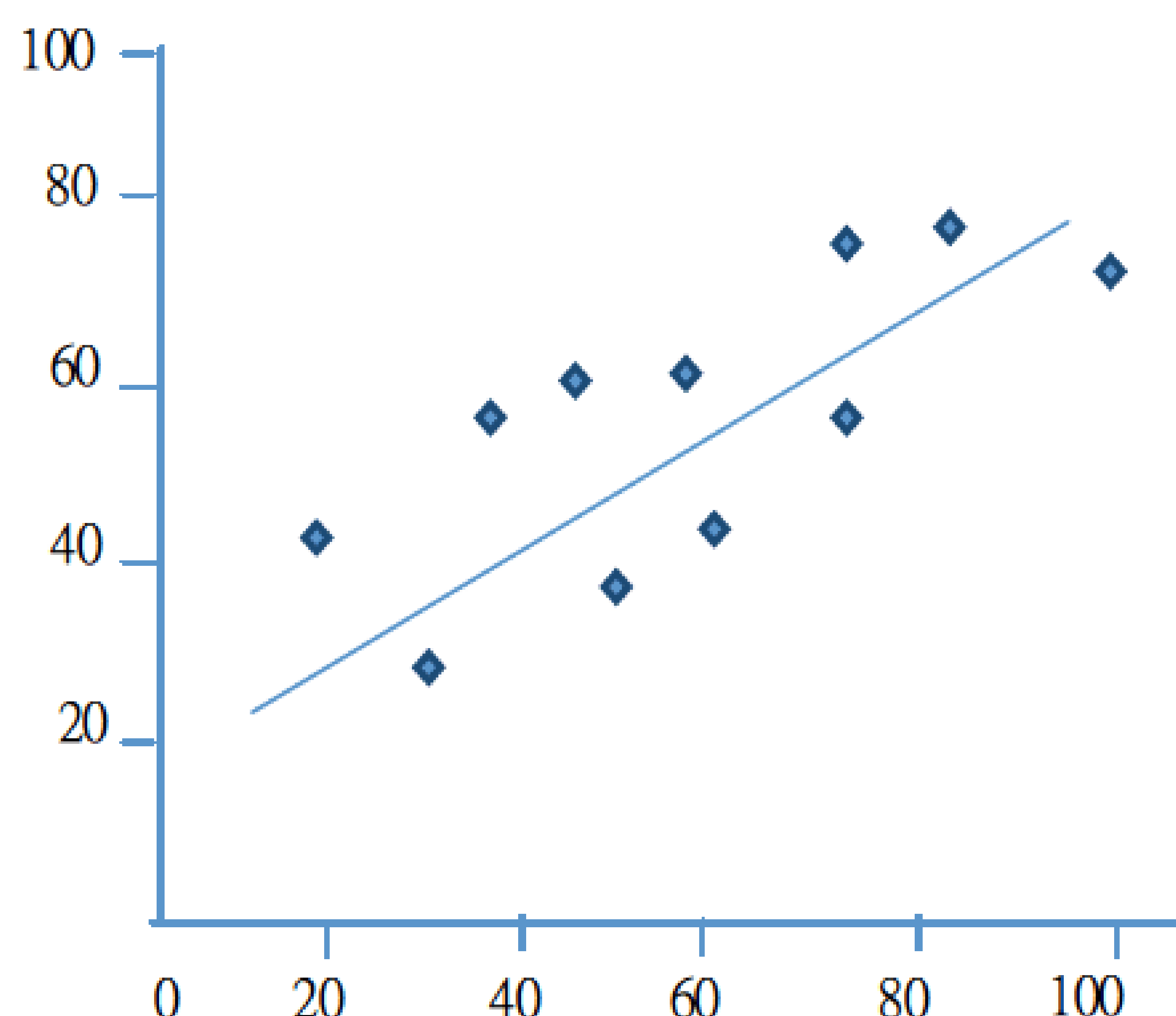


图 14-1　年龄与疾病的关系

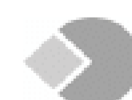
在线性回归中“因变量”是连续变项。如果 “因变量”不是连续变项，而是二分变项，就必须使用逻辑回归分析了。在逻辑回归分析中，我们可以将线性回归的公式：

$$y = b_0 + b_1 x$$

转换为 sigmoid 的函数，来帮助界定某个 data 的类。该函数如下：

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$

通过 sigmoid 计算出来的值 p（probability）如果大于 0.5，就归类为“会得病”；反之，归类为“不会得病”。逻辑回归的图形示例如图 14-2 所示。



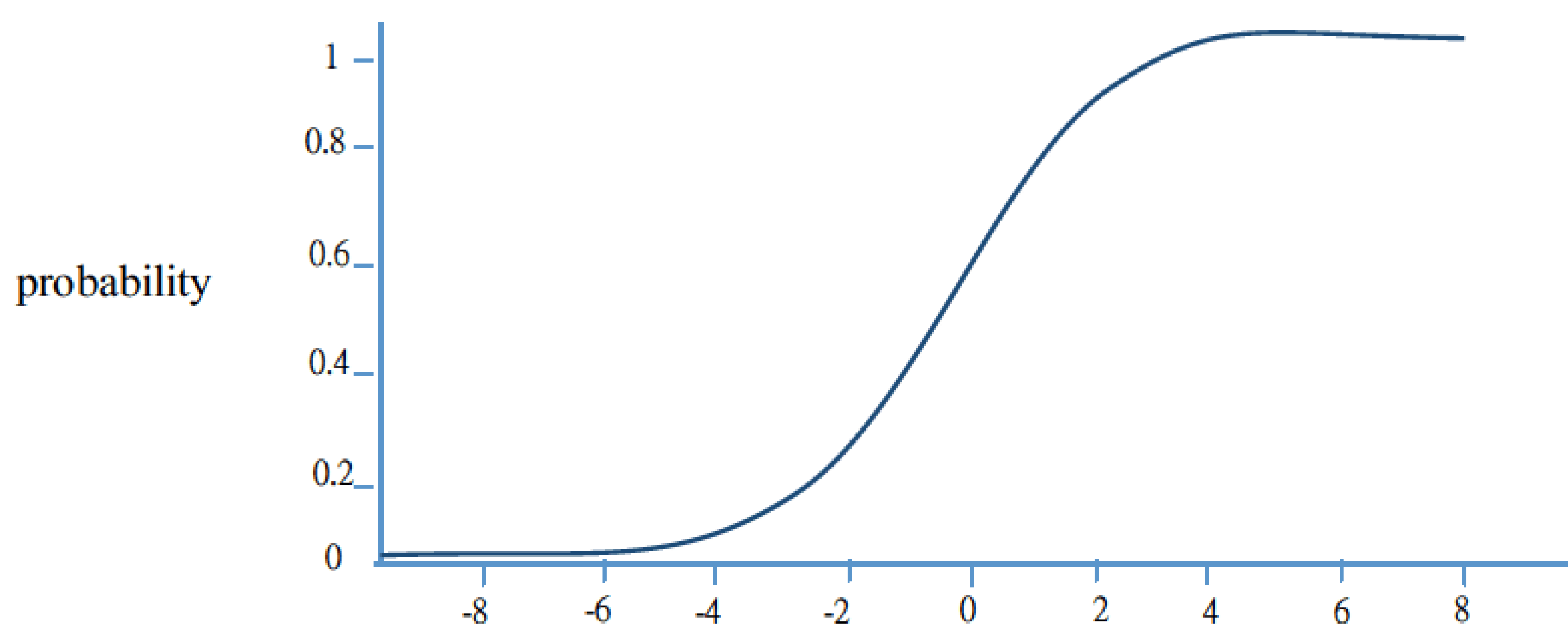


图 14-2　线性回归的图形示例

➢ **多元回归分析（Multiple Regression Analysis，或称为复回归分析）**

以上是简单线性回归，只使用一个自变量 x，多元回归分析使用多个自变量，公式如下：

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_px_p$$

可将其转换为 sigmoid 的函数。logistic regression 多元回归分析的公式如下：

$$p = \frac{1}{1 + e^{-(b_0 + b_1x_1 + b_2x_2 + \cdots + b_px_p)}}$$

14.2 RunLogisticRegressionWithSGDBinary.py 程序说明

我们已经使用 Spark MLlib 创建逻辑回归分析 RunLogisticRegressionWithSGDBinary.py，完整的程序代码可参考本书的范例程序。程序的架构与之前第 13 章介绍的 RunDecisionTreeBinary.py 类似，可以参考第 13 章的详细说明。本章仅说明 RunLogisticRegressionWithSGDBinary.py 重要的修改部分。

步骤 01 修改导入 LogisticRegression 链接库（见图 14-3）

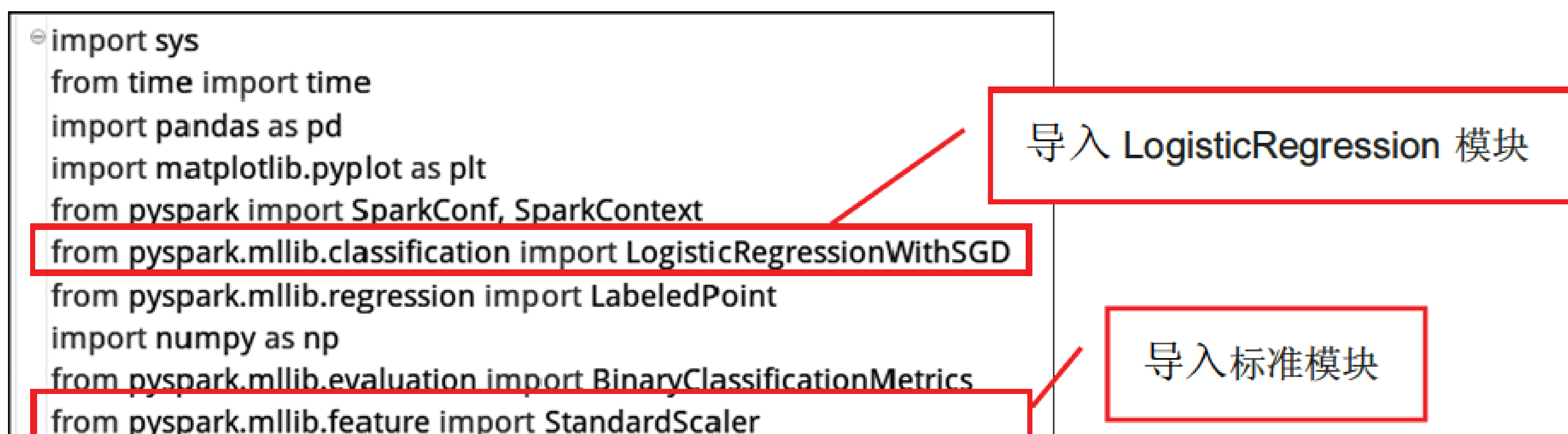


图 14-3　修改导入 LogisticRegression 链接库

步骤02 修改 PrepareData()

我们仍将使用第 13 章 StumbleUpon 数据集来执行逻辑回归分析二元分类。

下列程序基本上与第 13 章决策树的数据准备 PrepareData()类似。唯一不同的是，当我们进行回归分析算法时必须将数值特征（Numerical Features）字段进行标准化。因为数值特征字段单位不同，数字差异很大，所以彼此无法比较。这时就要使用标准化让数值特征字段具有共同的标准。

修改 PrepareData()，加入进行数据标准化的程序代码，如图 14-4 所示。

```
def PrepareData(sc):
    #---------------------1. 导入并转换数据 ------------
    print("开始导入数据...")
    rawDataWithHeader = sc.textFile(Path+"data/train.tsv")
    header = rawDataWithHeader.first()
    rawData = rawDataWithHeader.filter(lambda x:x !=header)
    rData=rawData.map(lambda x: x.replace("\"", ""))
    lines = rData.map(lambda x: x.split("\t"))
    print("共计：" + str(lines.count()) + "项")
    #---------------------2. 建立训练评估所需数据 RDD[LabeledPoint]------------
    print " 标准化之前：",
    categoriesMap = lines.map(lambda fields: fields[3]). \
                                    distinct().zipWithIndex().collectAsMap()
    labelRDD = lines.map(lambda r:  extract_label(r))
    featureRDD = lines.map(lambda r:  extract_features(r,categoriesMap,len(r) - 1))
    for i in featureRDD.first():
        print (str(i)+","),
    print ""
    print " 标准化之后：",
    stdScaler = StandardScaler(withMean=True, withStd=True).fit(featureRDD)
    ScalerFeatureRDD=stdScaler.transform(featureRDD)
    for i in ScalerFeatureRDD.first():
        print (str(i)+","),
    labelpoint=labelRDD.zip(ScalerFeatureRDD)
    labelpointRDD=labelpoint.map(lambda r: LabeledPoint(r[0], r[1]))
```

进行数据标准化之前

进行数据标准化之后

图 14-4　修改 PrepareData()

以上程序建立训练评估所需的数据主要分为以下两部分。

- 进行数据标准化之前：步骤 3 中介绍。
- 进行数据标准化之后：步骤 4 中介绍。

这是为了读者比较数据标准化的差异。

步骤03 spark-submit YARN-client 设置外部工具

标准化之前的程序代码与第 13.6 节决策树二元分类相同，可参考第 13.6 节的 PrepareDate()。为了能让读者了解标准化之前与之后数据的不同，我们以下列指令显示标准化之前的第一项数据。for 语句会读取 featureRDD.first() 的每一个字段，再使用 print (str(i)+",") 显示出每一个字段数据。

```
print " 标准化之前： ",
for i in featureRDD.first():
```

```
    print (str(i)+","),
```

步骤04　进行数据标准化

➢ 创建标准化刻度

使用 StandardScaler 创建标准化刻度 stdScaler，传入 withMean 与 withStd 参数，并且使用 fit 方法传入 featuresData 特征字段。

```
stdScaler = StandardScaler(withMean=True, withStd=True).fit(featureRDD)
```

➢ 进行标准化转换

使用上一条命令创建的标准化刻度 stdScaler 的 transform 方法传入 features 特征字段参数，进行标准化转换。

```
ScalerFeatureRDD=stdScaler.transform(featureRDD)
```

➢ 显示标准化之后的第一项数据

我们以下列语句显示标准化之后的第一项数据。for 语句会读取 ScalerFeatureRDD. first() 的每一个字段，再使用 print (str(i)+",")显示出每一个字段数据。

```
print " 标准化之后：",
for i in ScalerFeatureRDD.first():
    print (str(i)+","),
```

➢ 将 label 与标准化后的特征字段结合

我们使用 zip 将 label 与标准化后的特征字段结合起来建立 labelpoint。

```
labelpoint=labelRDD.zip(ScalerFeatureRDD)
```

➢ 创建 LabeledPoint 数据

后续训练必须使用 LabeledPoint 数据，所以使用之前的 labelpoint 用 map 转换来建立 LabelPoint 数据格式。

```
labelpointRDD=labelpoint.map(lambda r: LabeledPoint(r[0], r[1]))
```

步骤05　训练模型

使用 LogisticRegressionWithSGD 进行逻辑回归分析训练，如图 14-5 所示。

```
def trainEvaluateModel(trainData,validationData,
                    numIterations, stepSize, miniBatchFraction):
    startTime = time()
    model = LogisticRegressionWithSGD.train(trainData,
                    numIterations, stepSize, miniBatchFraction)
    AUC = evaluateModel(model, validationData)
    duration = time() - startTime
    print    "训练评估：使用参数 " + \
          " numIterations="+str(numIterations) +\
          " stepSize="+str(stepSize) + \
          " miniBatchFraction="+str(miniBatchFraction) +\
          " 所需时间 ="+str(duration) + \
          " 结果AUC = " + str(AUC)
    return (AUC,duration, numIterations, stepSize, miniBatchFraction,model)
```

图 14-5 进行逻辑回归分析训练

在 LogisticRegressionWithSGD 中使用 Stochastic Gradient Descent（简称 SGD）梯度下降法求得最佳解，必须输入下列内容：

```
LogisticRegressionWithSGD.train(trainData, numIterations, stepSize,miniBatchFraction)
```

返回：LogisticRegressionModel。

参数说明如表 14-1 所示。

表 14-1 参数说明

参数	说明
trainData	输入的训练数据
numIterations	使用 SGD 迭代次数，默认为 100
stepSize	每次执行 SGD 迭代步长大小，默认为 1
miniBatchFraction	每次迭代参与计算的样本比例，数值在 0～1 之间，默认为 1

步骤 06 parametersEval 函数

在之前的章节中，我们使用如下命令进行训练会返回 model 训练完成的模型：

```
LogisticRegressionWithSGD.train
(trainData, numIterations,
    stepSize, miniBatchFraction)
```

在这里我们评估的参数共有 3 个，即 numIterations、stepSize、miniBatchFraction。我们希望知道不同的参数值对于准确率的影响以及执行所需要的时间。评估的方法是同时间只评估一个参数。详细内容请参考第 13 章的说明。编写 parametersEval 参数评估函数，如图 14-6 所示。

```
def parametersEval(trainData, validationData):
    print("----- 评估numIterations参数使用 ---------")
    evalParameter(trainData, validationData,"numIterations",
                  numIterationsList=[5, 15, 20, 60, 100],
                  stepSizeList=[10],
                  miniBatchFractionList=[1 ])
    print("----- 评估stepSize参数使用 ---------")
    evalParameter(trainData, validationData,"stepSize",
                  numIterationsList=[100],
                  stepSizeList=[10, 50, 100, 200],
                  miniBatchFractionList=[1])
    print("----- 评估miniBatchFraction参数使用 ---------")
    evalParameter(trainData, validationData,"miniBatchFraction",
                  numIterationsList=[100],
                  stepSizeList =[100],
                  miniBatchFractionList=[0.5, 0.8, 1 ])
```

图 14-6 编写 parametersEval 参数评估函数

步骤 07 main 函数

编写 main 函数，与第 13 章的 main 函数类似，只是参数要改为 LogisticRegressionWithSGD 训练评估所需参数（见图 14-7，可参考第 13.14.1 小节的说明）。

```
if __name__ == "__main__":
    print("RunLogisticRegressionWithSGDBinary")
    sc=CreateSparkContext()
    print("==========数据准备阶段===============")
    (trainData, validationData, testData, categoriesMap) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("==========训练评估阶段===============")
    (AUC,duration, numIterationsParm, stepSizeParm, miniBatchFractionParm,model)= \
        trainEvaluateModel(trainData, validationData, 15, 10, 0.5)
    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
    elif   (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
        print("-----所有参数训练评估找出最好的参数组合---------")
        model=evalAllParameter(trainData, validationData,
                        [3, 5, 10,15],
                        [10, 50, 100],
                         [0.5, 0.8, 1 ])
    print("==========测试阶段===============")
    auc = evaluateModel(model, testData)
    print("使用test Data测试最佳模型，结果AUC:" + str(auc))
    print("==========预测数据===============")
    PredictData(sc, model, categoriesMap)
```

图 14-7　编写 main 函数

14.3 运行 RunLogisticRegressionWithSGDBinary.py 进行参数评估

步骤 01 运行外部程序（见图 14-8）

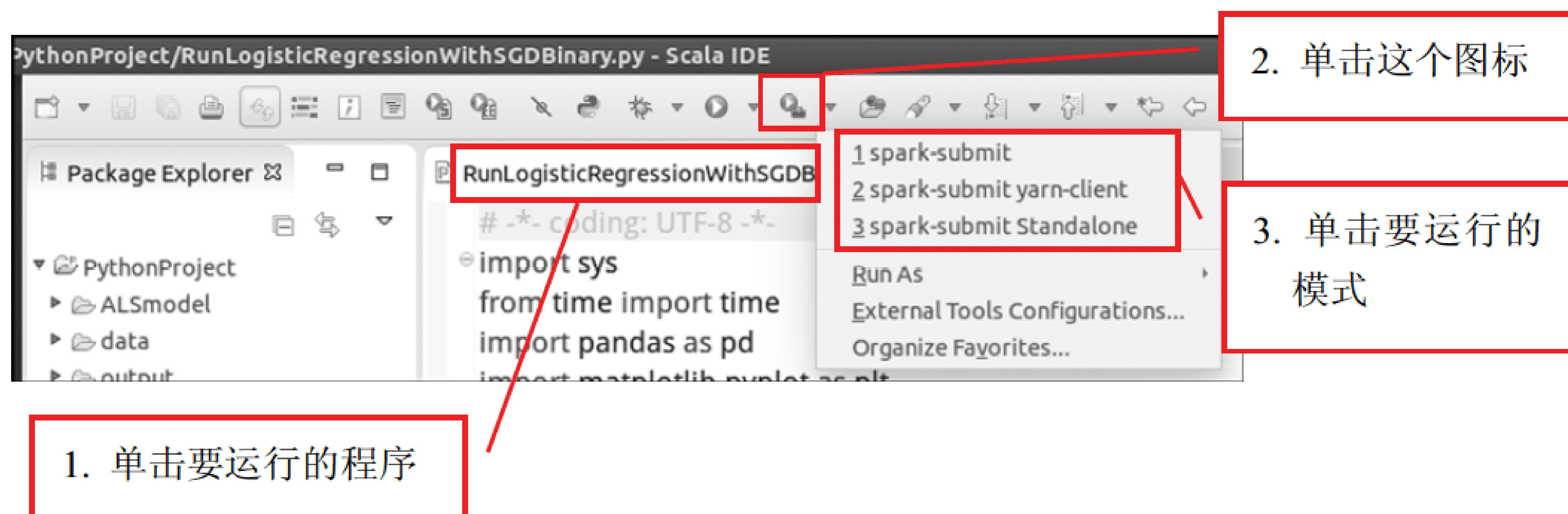


图 14-8　运行外部程序

步骤02 输入参数“-e”（见图 14-9）



图 14-9 输入参数“-e”

步骤03 数据准备阶段

我们可以看到数据在标准化前后的数据（见图 14-10）。

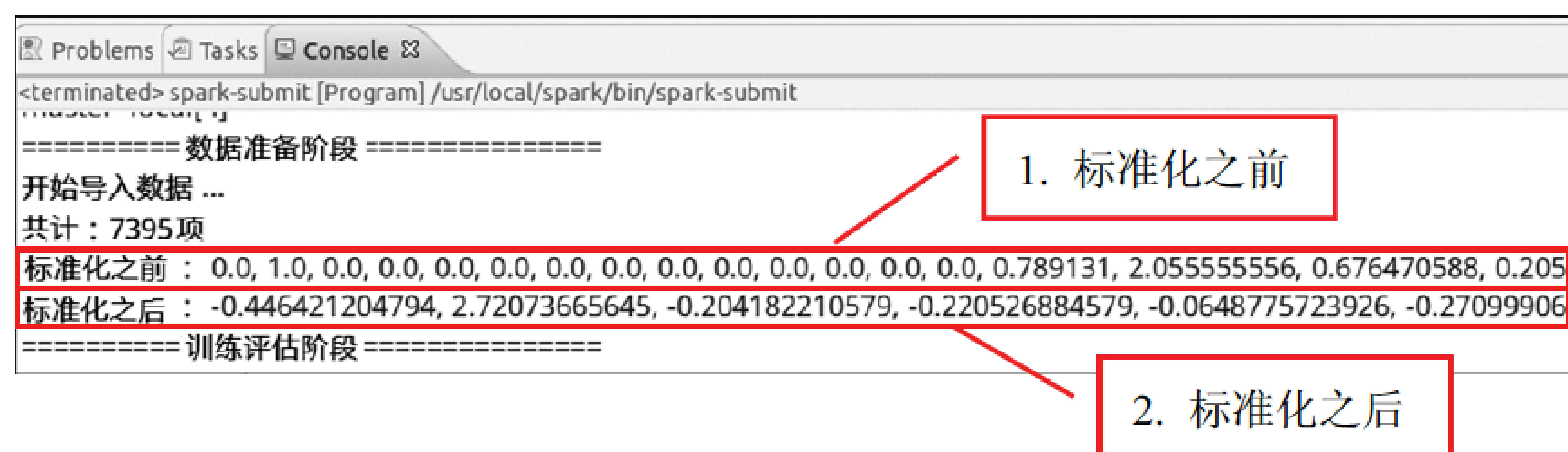


图 14-10 数据准备阶段

从中可以看到标准化之前与之后的差异。

步骤04 numIterations 参数评估

接下来，程序会显示如图 14-11 所示的 numIterations 参数评估结果图。其中，柱形图代表 AUC、折线图代表运行的时间。从图 14-11 可以看到 numIterations=15 时 AUC 最高，而 numIterations=60 时运行时间最多。

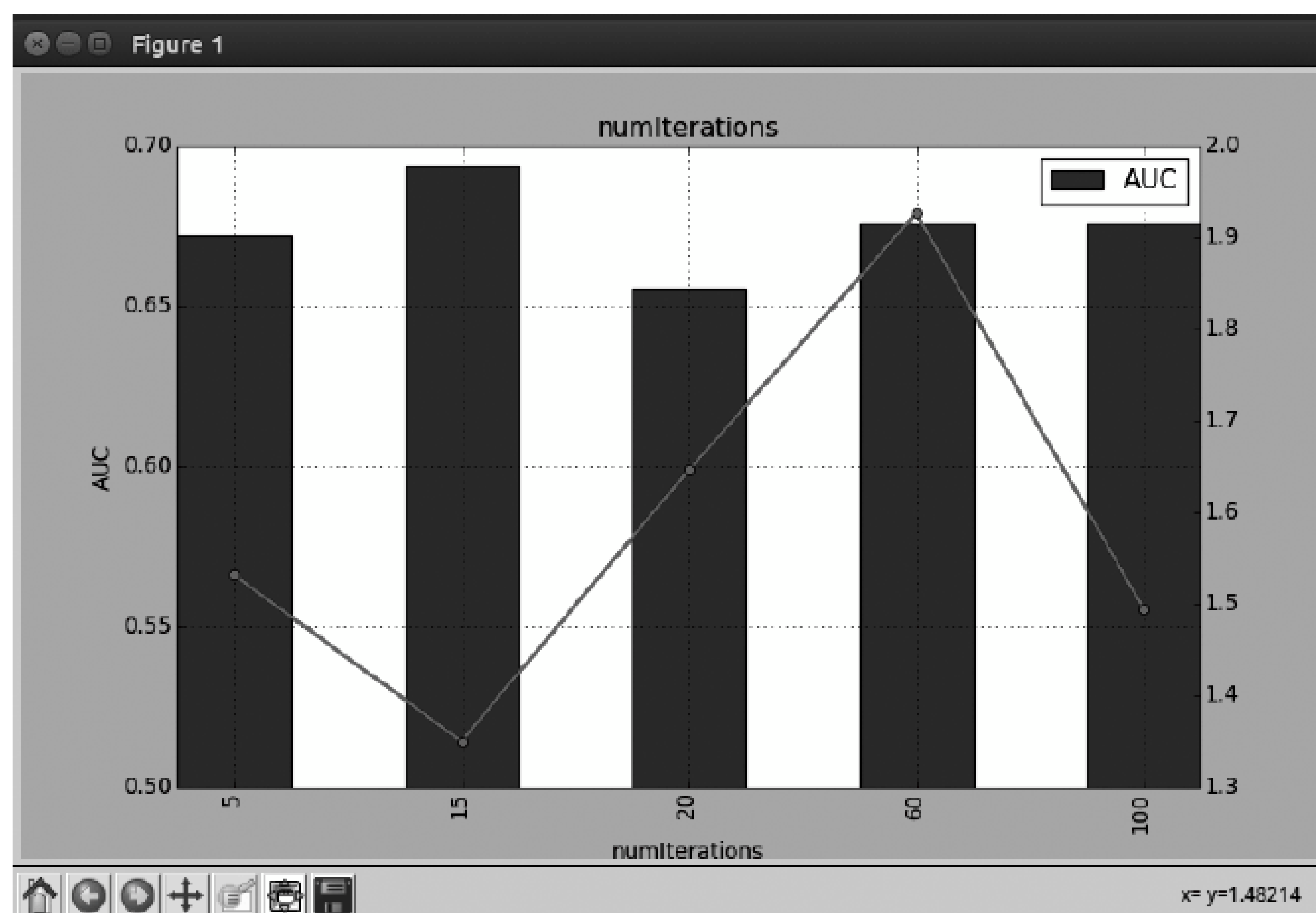
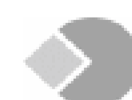
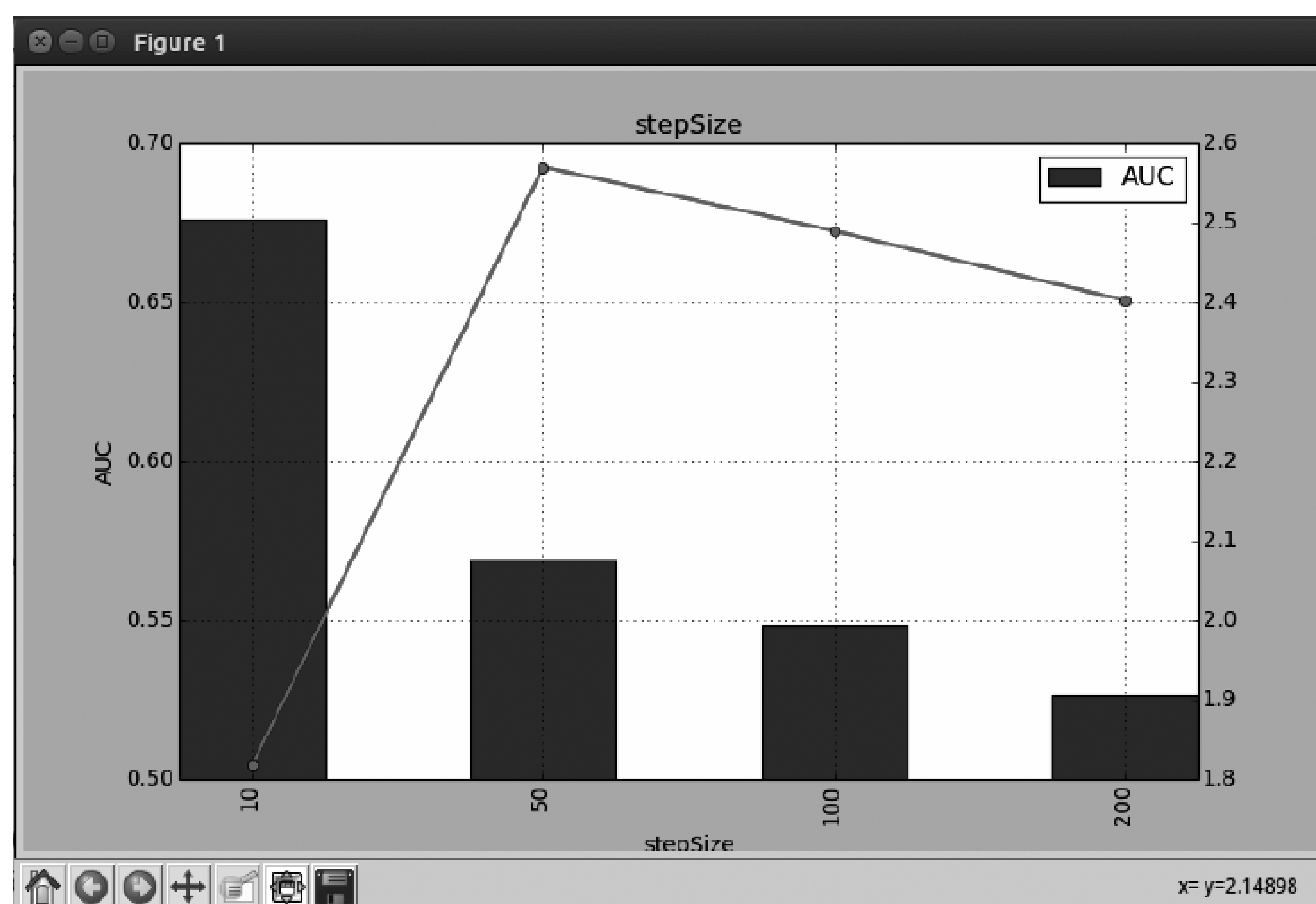


图 14-11 numIterations 参数评估结果图



步骤 05　stepSize 参数评估

从图 14-12 可以看到，stepSize=10 时 AUC 最高。stepSize 越大，所需时间越少。



14-12　stepSize 参数评估

步骤 06　miniBatchFraction 参数评估

从图 14-13 可以看到，miniBatchFraction=1 时 AUC 最高。所需的时间没有太大的差异。

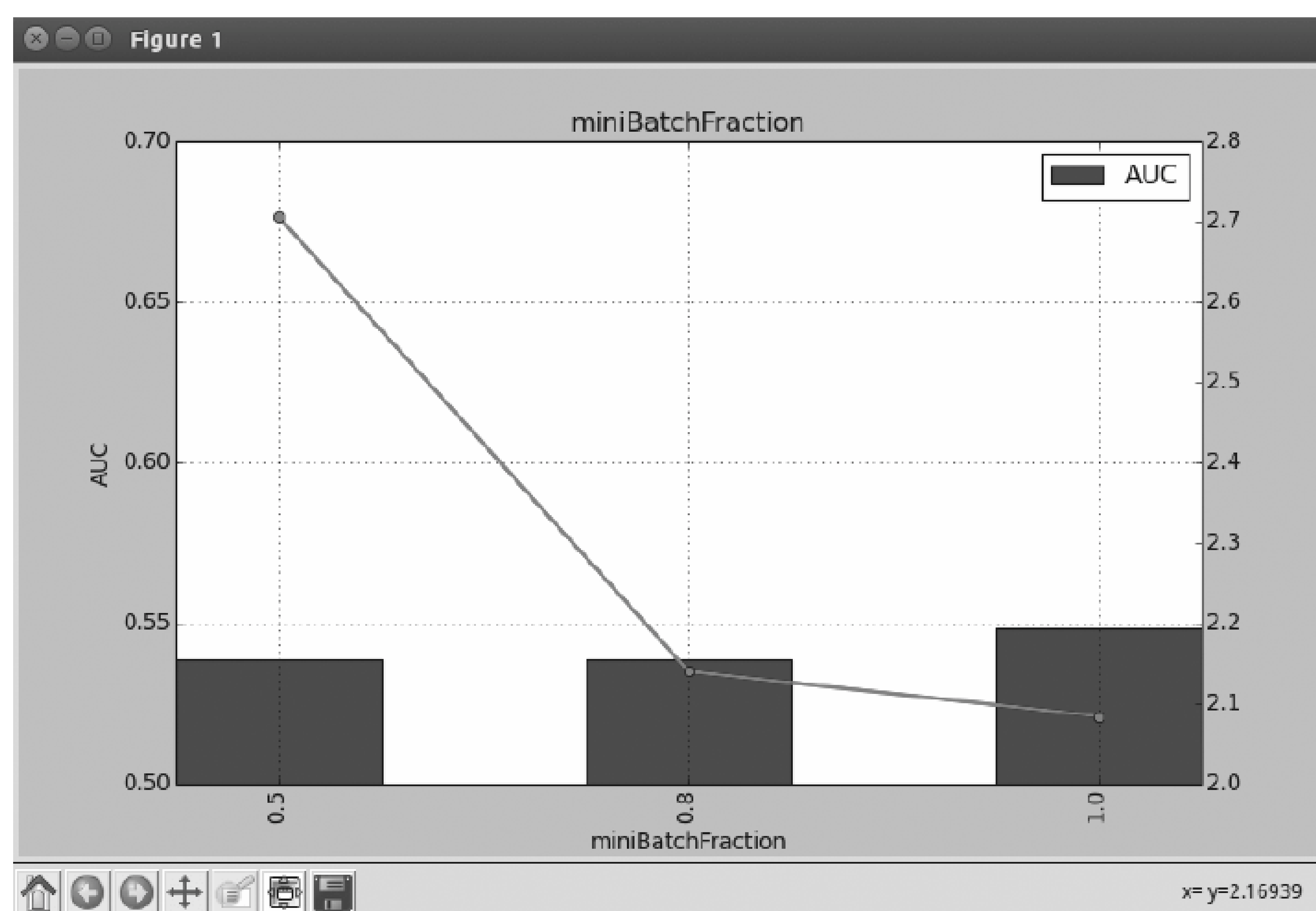


图 14-13　miniBatchFraction 参数评估

14.4 找出最佳参数组合

步骤01 运行程序，输入参数“-a”（见图 14-14）



图 14-14 输入参数

步骤02 训练评估参数，找出最佳参数组合（见图 14-15）

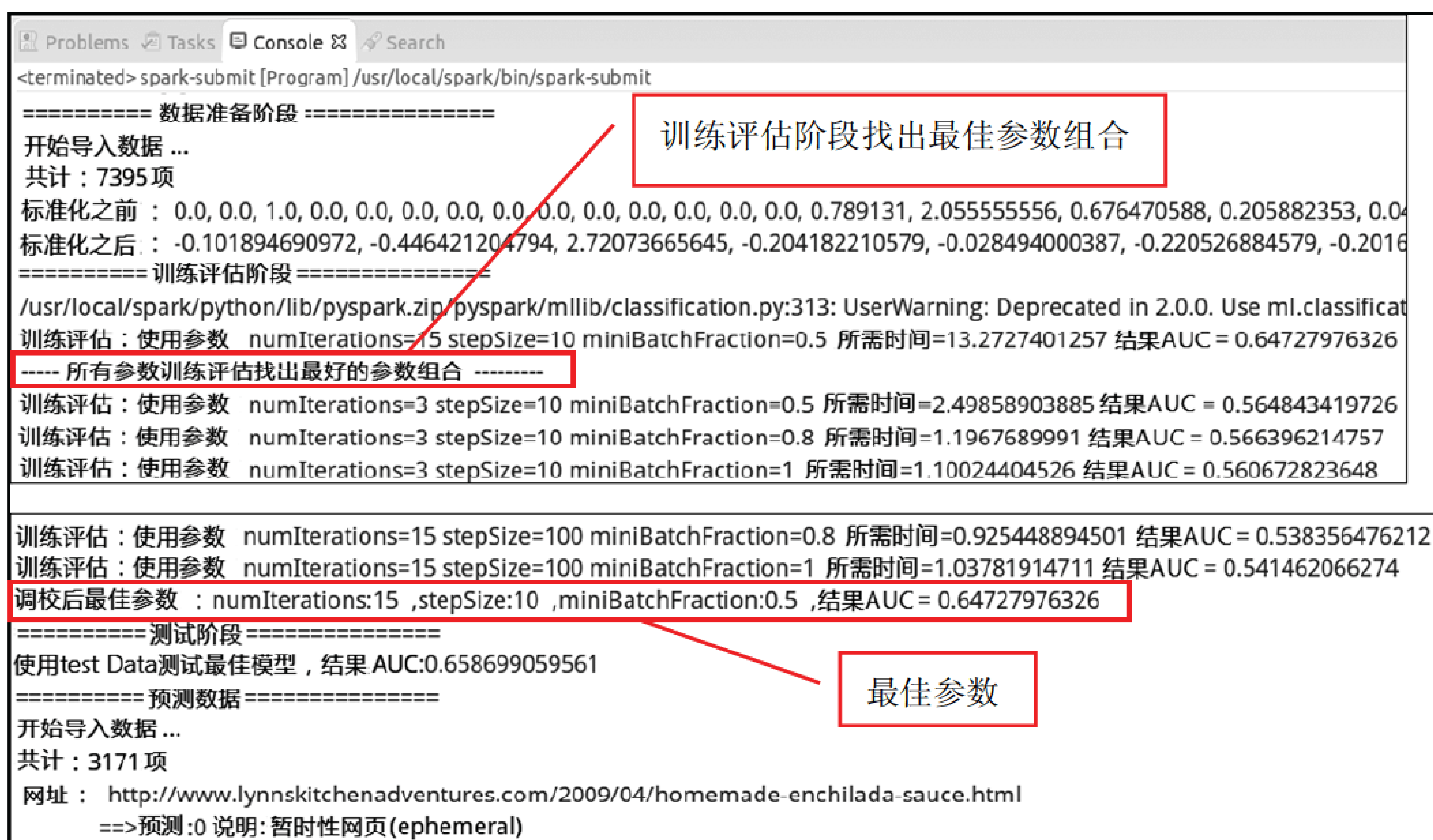


图 14-15 找出最佳参数组合

14.5 修改程序使用参数进行预测

因为训练评估比较费时，所以当我们找出最佳参数组合时，可以修改 trainEvaluateModel 程序，改为最佳参数组合。下次要运行预测程序时可以不再运行训练评估，直接使用最佳参数进行预测。

步骤 01　修改默认的训练参数为最佳参数组合

修改 trainEvaluateModel 程序，改为最佳参数组合，如图 14-16 所示。

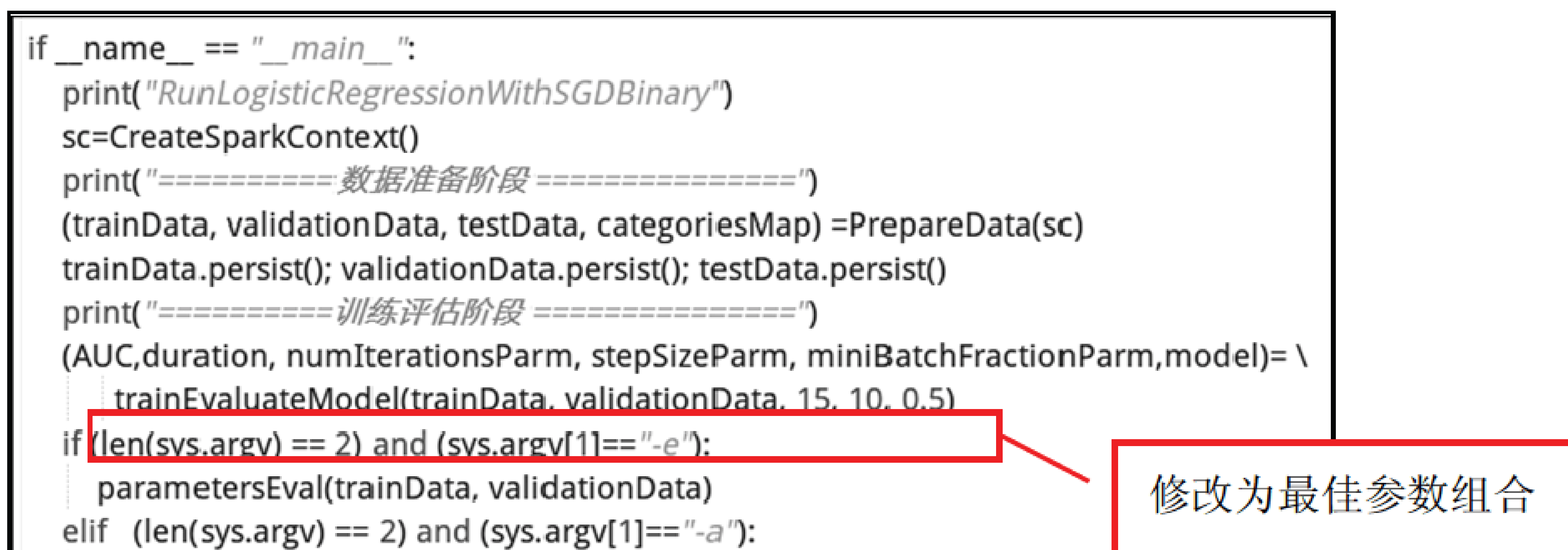


图 14-16　修改为最佳参数组合

步骤 02　运行外部程序（见图 14-17）

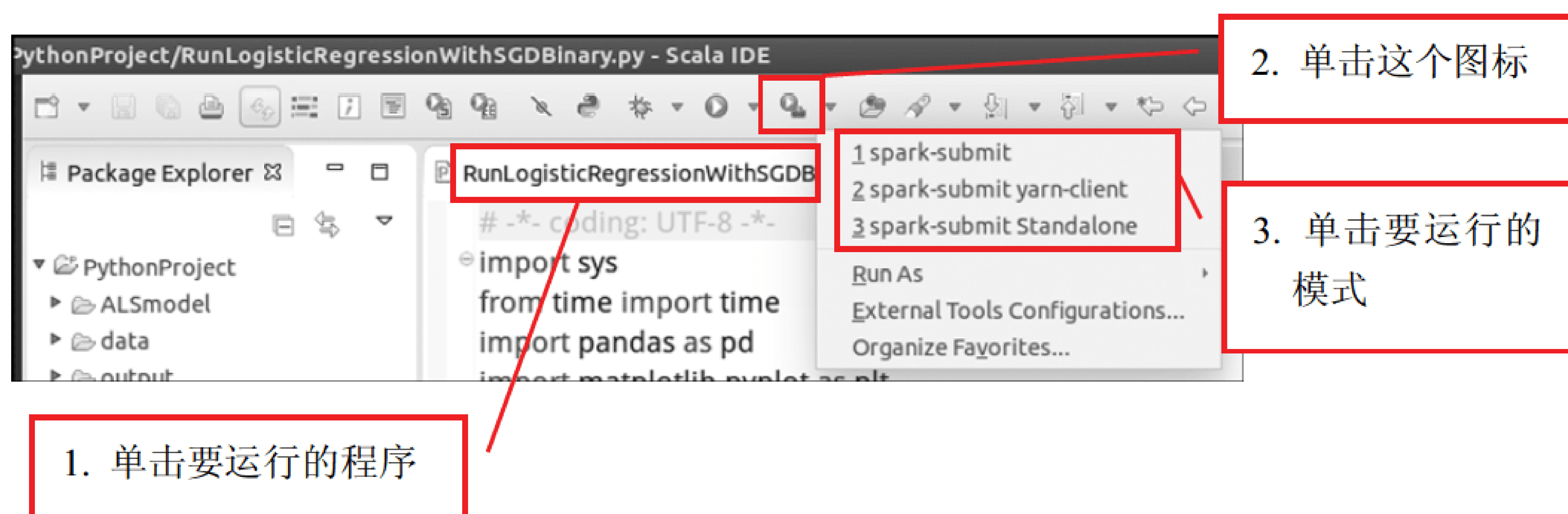


图 14-17　运行外部程序

步骤 03　运行 RunLogisticRegressionWithSGD.py，不要输入参数（见图 14-18）



图 14-18　不输入参数

步骤 04　再次运行程序

运行结果如图 14-19 所示。从中可以看出，程序训练完成后就可以进行预测。因为不进行训练评估，所以运行所需时间较少。

```
Problems  Tasks  Console
<terminated> spark-submit parameter [Program] /usr/local/spark/bin/spark-submit
========== 数据准备阶段 ===============
开始导入数据 ...
共计：7395项
标准化之前 ： 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.789131, 2.055555556, 0.676470588, 0.205882353, 0.047058
标准化之后 ： -0.446421204794, 2.72073665645, -0.204182210579, -0.220526884579, -0.0648775723926, -0.270999069693, -0.6807527
==========训练评估阶段 ===============
训练评估：使用参数  numIterations=15 stepSize=10 miniBatchFraction=0.5 所需时间=8.06712317467 结果AUC = 0.680571903574
==========测试阶段===============
使用test Data测试最佳模型，结果 AUC:0.648458005249
==========预测数据===============
开始导入数据 ...
共计：3171 项
 网址：  http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html
        ==>预测:0 说明: 暂时性网页(ephemeral)

 网址：  http://lolpics.se/18552-stun-grenade-ar
        ==>预测:0 说明: 暂时性网页(ephemeral)

 网址：  http://www.xcelerationfitness.com/treadmills.html
        ==>预测:0 说明: 暂时性网页(ephemeral)

 网址：  http://www.bloomberg.com/news/2012-02-06/syria-s-assad-deploys-tactics-of-father-to-crush-revolt-threatening-reign.html
        ==>预测:0 说明: 暂时性网页(ephemeral)
```

1. 数据准备

2. 训练评估阶段

3. 训练阶段

4. 预测数据

图 14-19　运行程序结果

（1）数据准备阶段：显示 trainData、validationData、testData 导入项数。

（2）训练评估阶段：执行最佳参数组合 numIterations = 15、stepSize = 10、miniBatchFraction = 0.5，AUC 大约为 0.68。

（3）测试阶段：使用测试数据再测试模型，AUC 大约为 0.64。与训练评估阶段差异不大，确认没有 overfitting 的问题。

（4）预测阶段：使用 test.tsv 预测是长青网页还是暂时性网页。预测的数据会显示网址，可以使用浏览器查看一下是否准确。

14.6 结论

本章介绍了 Spark MLlib 逻辑回归二元分类的基本原理，完成了数据准备、训练模型、预测网页是暂时性还是长青的，并且通过训练评估找出最佳参数组合，提高预测准确率。下一章我们将介绍支持向量机 SVM 二元分类。

第 15 章 Python Spark MLlib 支持向量机SVM二元分类

支持向量机（Support Vector Machine，SVM）是一种监督式学习的方法，广泛运用于机器学习分类。

本章将使用第 13 章介绍的 StumbleUpon 数据集，运用支持向量机 SVM 二元分类来预测网页是暂时性的还是长青的，并且通过训练评估找出最佳参数组合，提高预测准确度。

15.1 支持向量机 SVM 算法的基本概念

SVM 可以用来作为分类（Classification）的工具。在说明 SVM 分类的方法之前，我们先看图 15-1。图 15-1 中有圆点以及三角形的点，希望找出一条线段进行比较好的分类。

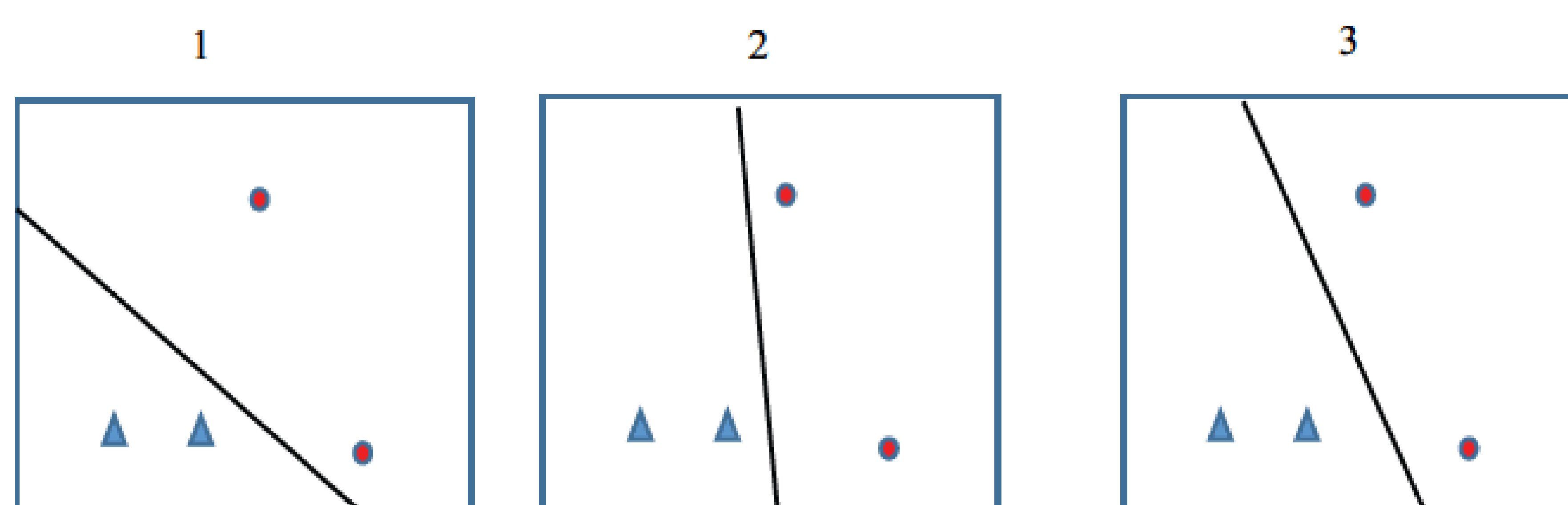


图 15-1　不同分类的示意图

SVM 主要是在寻找最大边际的超平面（Maximum Marginal Hyperplane），因为其具有较高的分类准确性，并且有较高的误差容忍度。如图 15-2 所示的虚线范围，其中的第 3 张图的分类具有最大的边际区，所以对于 SVM 算法而言，此图是比较好的分类。

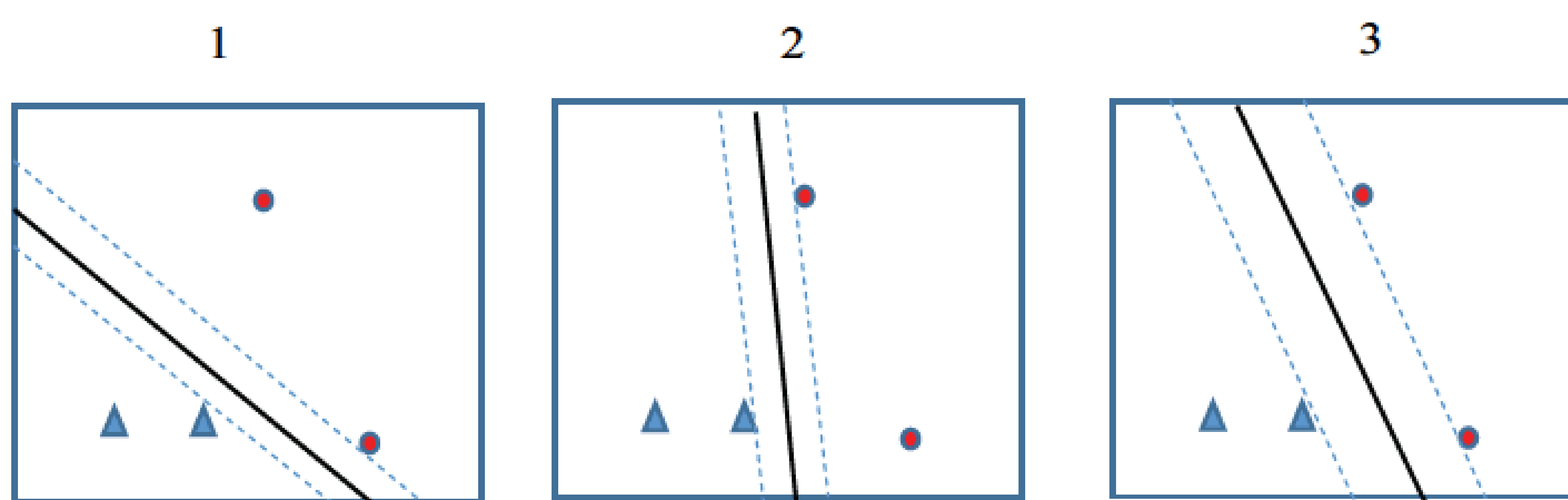


图 15-2　寻找最大边际的超平面

➤ **RunSVMWithSGDBinary.py 程序说明**

我们已经使用 Spark MLlib 创建了支持向量机分析 RunSVMWithSGDBinary.py，完整的程序代码可参考本书范例程序。程序的架构与之前第 13 章介绍的 RunDecisionTreeBinary.py 类似，可以参考第 13 章的详细说明。以下仅说明 RunSVMWithSGDBinary.py 重要的修改部分。

步骤 01 导入 SVMWithSGD 链接库（见图 15-3）

```
import sys
from time import time
import pandas as pd
import matplotlib.pyplot as plt
from pyspark import SparkConf, SparkContext
from pyspark.mllib.classification import SVMWithSGD
from pyspark.mllib.regression import LabeledPoint
import numpy as np
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.feature import StandardScaler
```

导入 SVMWithSGD 模块

图 15-3　导入 SVMWithSGD 模块

步骤 02 修改 PrepareData()以便加入数据标准化的程序代码

支持向量机 SVM 二元分类，也是需要进行数据标准化，这部分请参考第 14 章加入数据标准化的部分。

步骤 03 trainModel 训练模型

接下来使用 SVMWithSGD 进行逻辑回归分析训练，如图 15-4 所示。

```
def trainEvaluateModel(trainData,validationData,
                       numIterations, stepSize, regParam):
    startTime = time()
    model = SVMWithSGD.train(trainData, numIterations, stepSize, regParam)
    AUC = evaluateModel(model, validationData)
    duration = time() - startTime
    print    " 训练评估：使用参数 " + \
             " numIterations="+str(numIterations) +\
             " stepSize="+str(stepSize) + \
             " regParam="+str(regParam) +\
             " 所需时间="+str(duration) + \
             " 结果AUC = " + str(AUC)
    return (AUC,duration, numIterations, stepSize, regParam,model)
```

图 15-4　使用 SVMWithSGD 进行逻辑回归分析训练

在 SVMWithSGD 使用 Stochastic Gradient Descent（简称 SGD）梯度下降法方式求得最佳解，所以必须输入下列参数：

```
SVMWithSGD.train(trainData, numIterations, stepSize, regParam)LogisticRegressionModel
```

返回：LogisticRegressionModel。

参数	说明
trainData	输入的训练数据 LabeledPointRDD
numIterations	使用 SGD 迭代次数，默认为 100
stepSize	每次执行 SGD 迭代步长的大小，默认为 1
regParam	正则化参数，数值为 0～1 之间

步骤 04 输入 parametersTunning 函数（见图 15-5）

```
def parametersEval(trainData, validationData):
    print("----- 评估numIterations参数使用 ---------")
    evalParameter(trainData, validationData,"numIterations",
                  numIterationsList= [1, 3, 5, 15, 25],
                  stepSizeList=[100],
                  regParamList=[1 ])
    print("----- 评估stepSize参数使用 ---------")
    evalParameter(trainData, validationData,"stepSize",
                  numIterationsList=[25],
                  stepSizeList= [10, 50, 100, 200],
                  regParamList=[1])
    print("----- 评估regParam参数使用 ---------")
    evalParameter(trainData, validationData,"regParam",
                  numIterationsList=[25],
                  stepSizeList =[100],
                  regParamList=[0.01, 0.1, 1 ])
```

图 15-5　输入 parametersTunning 函数

在这里评估的参数共有 3 个：numIterations、stepSize、regParam。我们希望得知不同的参数值对准确率的影响以及运行所需要的时间。评估的方法是同时间只评估一个参数。

步骤 05 main 函数

编写 main 函数（见图 15-6），与第 13 章的 main 函数类似，只是参数改为 SVMWithSGD 训练评估所需的参数。请参考第 13.14 节的说明。

```
if __name__ == "__main__":
    print("RunSVMWithSGDBinary")
    sc=CreateSparkContext()
    print("==========数据准备阶段 ===============")
    (trainData, validationData, testData, categoriesMap) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("==========训练评估阶段 ===============")
    (AUC,duration, numIterations, stepSize, regParam,model)= \
        trainEvaluateModel(trainData, validationData, 3, 50, 1)
    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
    elif  (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
        print("-----所有参数训练评估找出最好的参数组合 ---------")
        model=evalAllParameter(trainData, validationData,
                        [1, 3, 5, 15, 25],
                        [10, 50, 100, 200],
                        [0.01, 0.1, 1 ])
    print("==========测试阶段 ===============")
    auc = evaluateModel(model, testData)
    print("使用test Data测试最佳模型，结果AUC:" + str(auc))
    print("==========预测数据 ===============")
    PredictData(sc, model, categoriesMap)
```

图 15-6　编写 main 函数

15.2 运行 SVMWithSGD.py 进行参数评估

步骤 01 运行外部程序（见图 15-7）

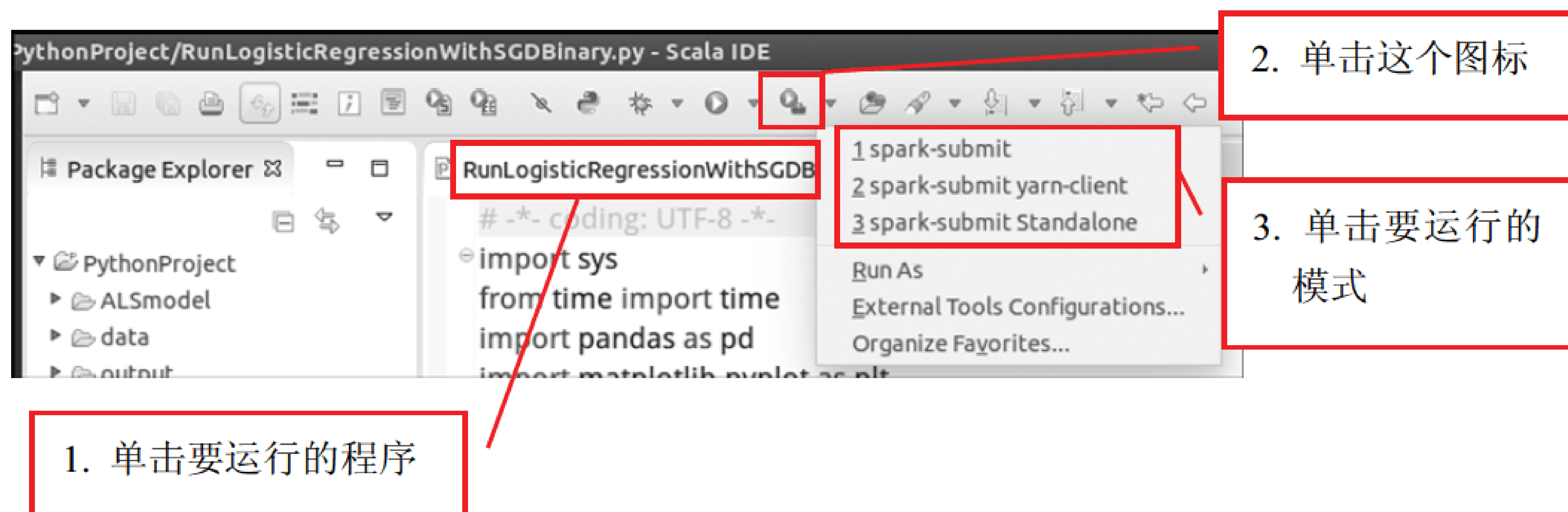
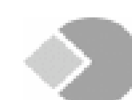


图 15-7　运行外部程序



步骤 02　输入参数 “-e”（见图 15-8）



图 15-8　输入参数 “-e”

步骤 03　numIterations 参数评估

从图 15-9 可以看到 numIterations=1 时 AUC 最高，不过差异不大。除了 numIterations=1 需要较多时间之外，其余随着 numIterations 增大所需的时间越来越多。

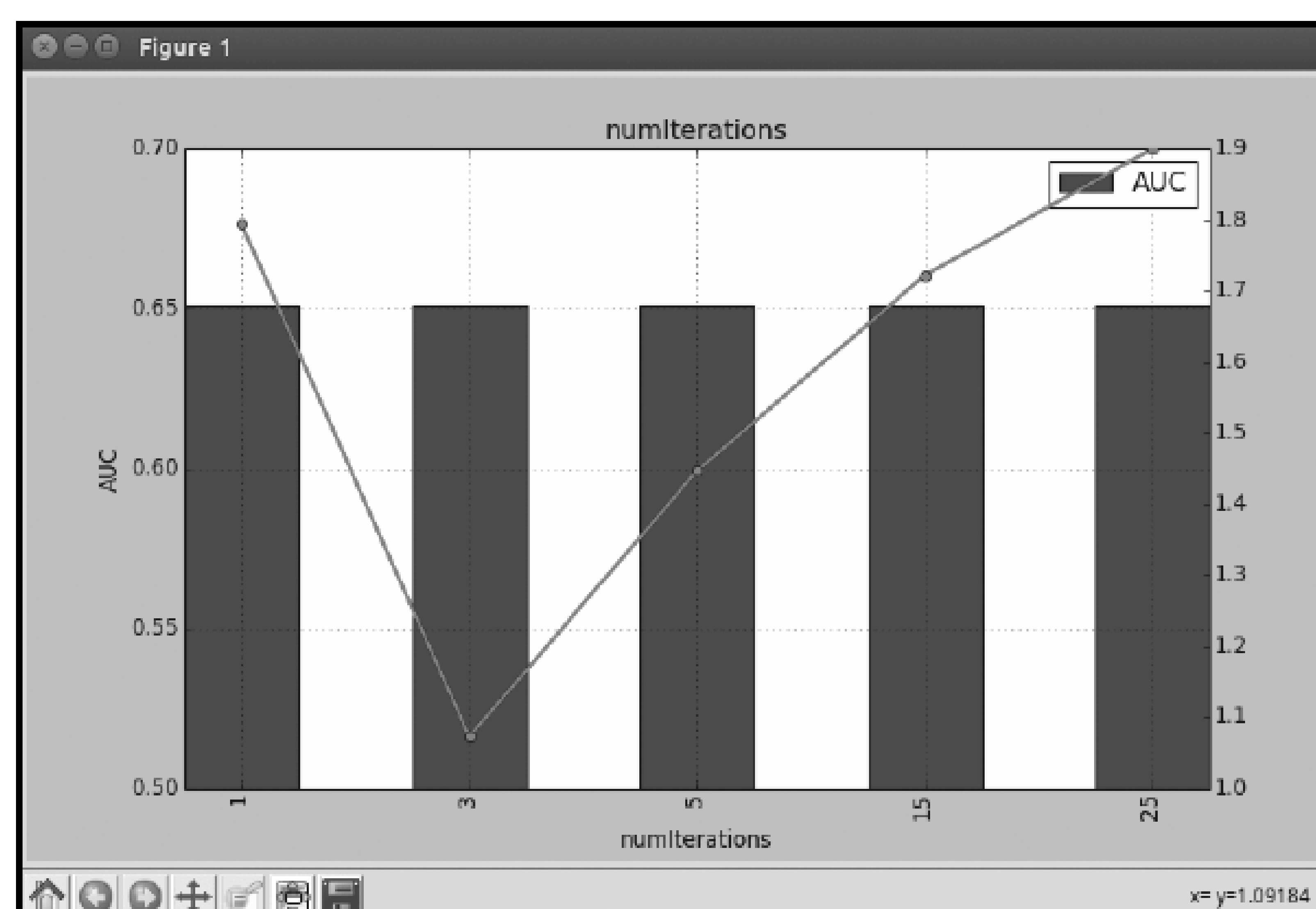


图 15-9　numIterations 参数评估

步骤 04　stepSize 参数评估

从图 15-10 可以看到 stepSize=200 时 AUC 最高。在 100 之前，stepSize 越大，所需的时间越少。

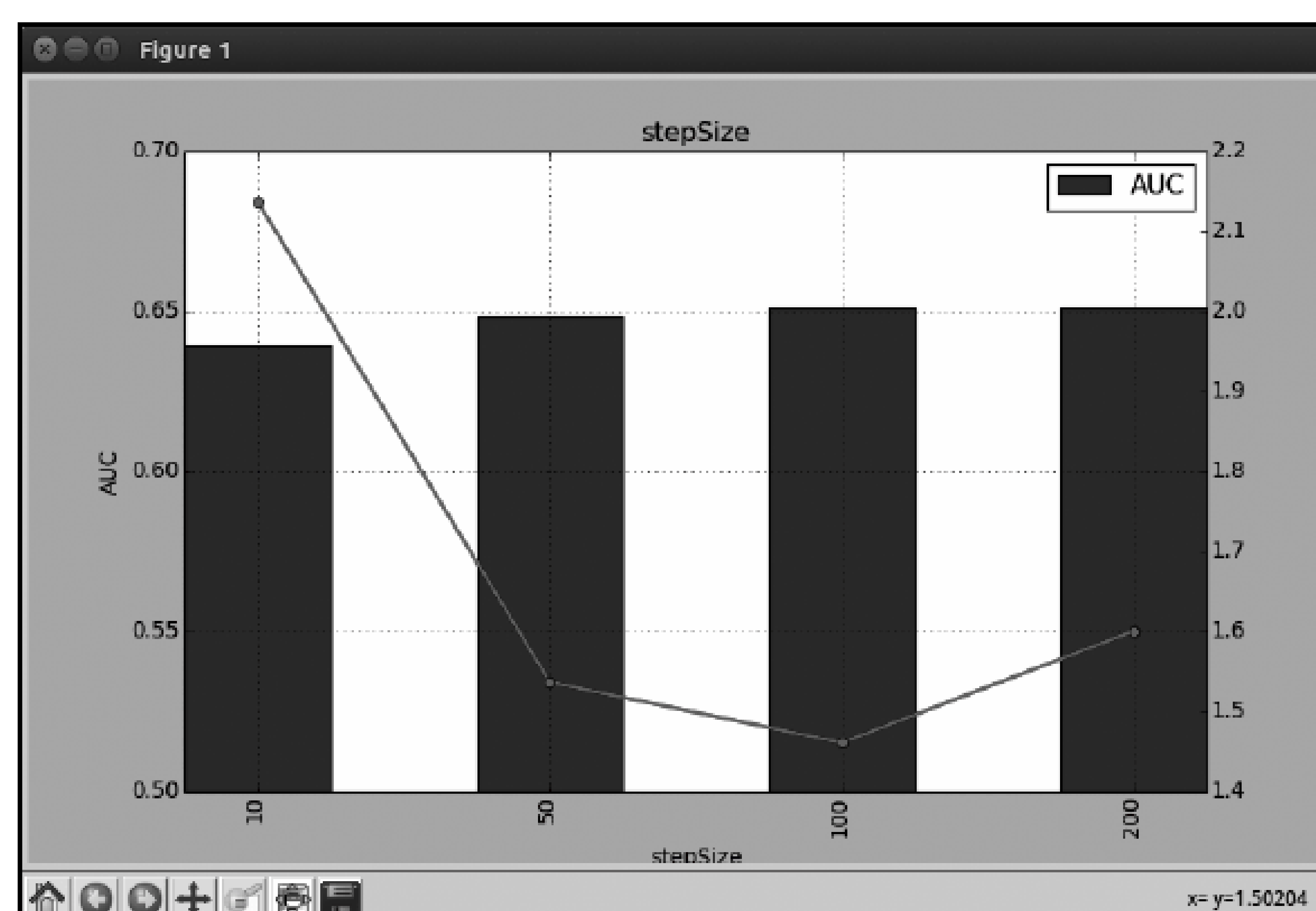


图 15-10　stepSize 参数评估

步骤05 regParam 参数评估

从图 15-11 可以看到 regParam=1 时 AUC 最高。regParam 越大，所需时间越少。

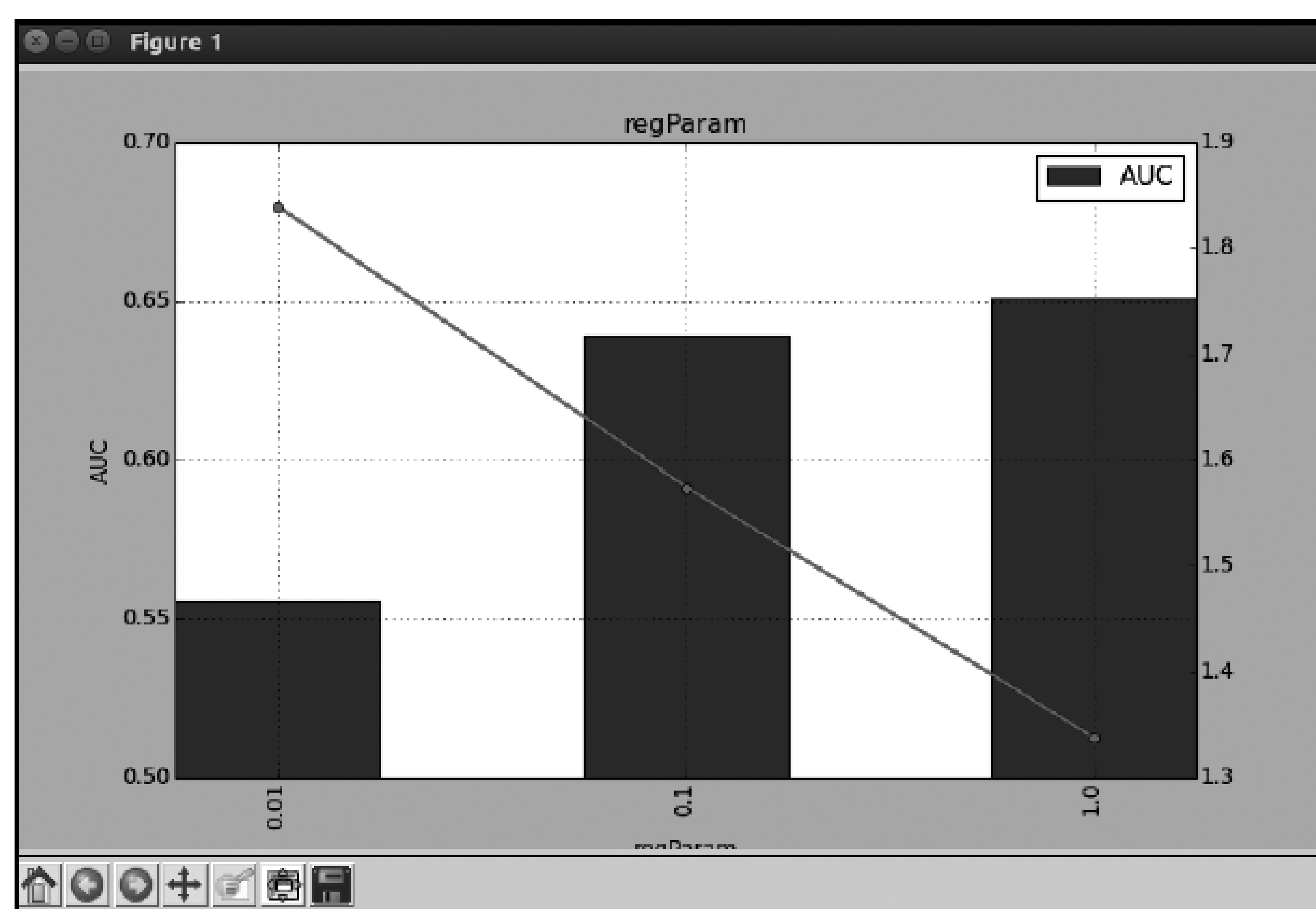


图 15-11　regParam 参数评估

15.3 运行 SVMWithSGD.py 训练评估参数并找出最佳参数组合

接下来，我们以 evaluateAllParameter 函数训练评估 3 种参数，希望找出最好的参数组合。

步骤01 输入参数“-a”（见图 15-12）



图 15-12　输入参数“-a”

步骤02 进行训练评估参数，找出最佳的参数组合（见图 15-13）

```
Problems  Tasks  Console  Search
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit
========== 数据准备阶段 ===============
开始导入数据 ...
共计：7395项
标准化之前 ： 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.789131, 2.055555556, 0.676470588, 0.2058823
标准化之后 ： -0.101894690972, -0.446421204794, 2.72073665645, -0.204182210579, -0.028494000387, -0.220526884579,
========== 训练评估阶段 ===============
训练评估：使用参数 numIterations=3 stepSize=50 regParam=1 所需时间=10.3397679329 结果AUC = 0.661123064969
----- 所有参数训练评估找出最好的参数组合 ----------
训练评估：使用参数  numIterations=1 stepSize=10 regParam=0.01 所需时间=1.59783101082 结果AUC = 0.665127607435
训练评估：使用参数  numIterations=1 stepSize=10 regParam=0.1 所需时间=1.8175110817 结果AUC = 0.665127607435
训练评估：使用参数  numIterations=1 stepSize=10 regParam=1 所需时间=0.893939971924 结果AUC = 0.665127607435

训练评估：使用参数 numIterations=25 stepSize=100 regParam=1 所需时间=1.10974192619 结果AUC = 0.665053438193
训练评估：使用参数 numIterations=25 stepSize=200 regParam=0.01 所需时间=1.08016705513 结果AUC= 0.566167574108
训练评估：使用参数 numIterations=25 stepSize=200 regParam=0.1 所需时间=1.04560422897 结果AUC= 0.665053438193
训练评估：使用参数 numIterations=25 stepSize=200 regParam=1 所需时间=1.05699706078 结果AUC= 0.665053438193
调校后最佳参数：numIterations:3 ,stepSize:50 ,regParam:1 ,结果AUC= 0.666490219802
==========测试阶段===============
使用test Data测试最佳模型，结果AUC :0.650280898876
==========预测数据===============
开始导入数据...
共计：3171项
```

训练评估阶段，找出最佳参数组合

最佳参数

图 15-13 找出最佳参数组合

15.4 运行 SVMWithSGD.py 使用最佳参数进行预测

训练评估比较费时，当我们已经找出最佳参数组合时可以修改 trainEvaluateModel 程序为最佳参数组合。下次要运行预测程序时，可以不再运行训练评估，直接使用最佳参数进行预测。

步骤 01 修改 trainEvaluateModel 为最佳参数组合

修改 trainEvaluateModel 程序，改为最佳参数组合，如图 15-14 所示。

```
*RunSVMWithSGDBinary
if __name__ == "__main__":
    print("RunSVMWithSGDBinary")
    sc=CreateSparkContext()
    print("==========数据准备阶段 ===============")
    (trainData, validationData, testData, categoriesMap) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("==========训练评估阶段 ===============")
    (AUC,duration, numIterations, stepSize, regParam,model)= \
        trainEvaluateModel(trainData, validationData, 3, 50, 1)
    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
    elif   (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
        print("-----所有参数训练评估找出最好的参数组合 ---------")
        model=evalAllParameter(trainData, validationData,
                        [1, 3, 5, 15, 25],
                        [10, 50, 100, 200],
                        [0.01, 0.1, 1 ])
    print("==========测试阶段 ===============")
    auc = evaluateModel(model, testData)
    print("使用test Data测试最佳模型，结果AUC:" + str(auc))
    print("==========预测数据 ===============")
    PredictData(sc, model, categoriesMap)
```

修改为最佳参数组合

图 15-14 修改为最佳参数组合

步骤02 运行外部程序（见图 15-15）

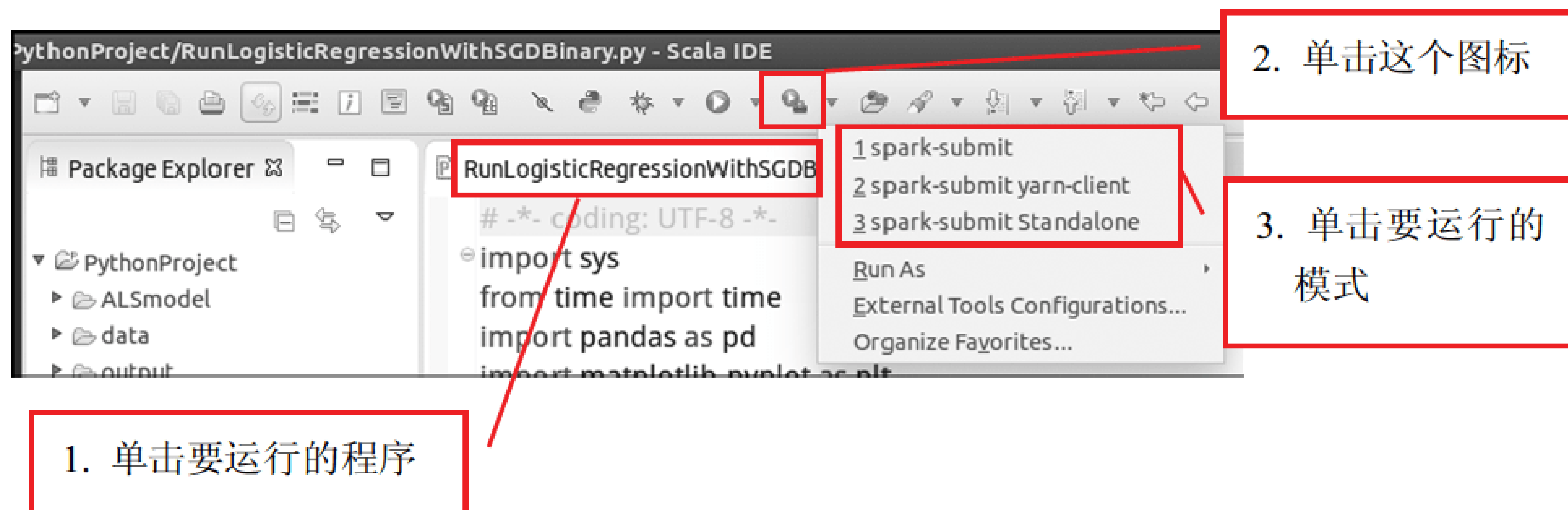


图 15-15　运行外部程序

步骤03 不输入参数（见图 15-16）



图 15-16　不输入参数

步骤04 运行结果

运行结果如图 15-17 所示，从中可以看到程序训练完成后就进行了预测。因为不需要进行训练评估，所以运行所需时间比较少。

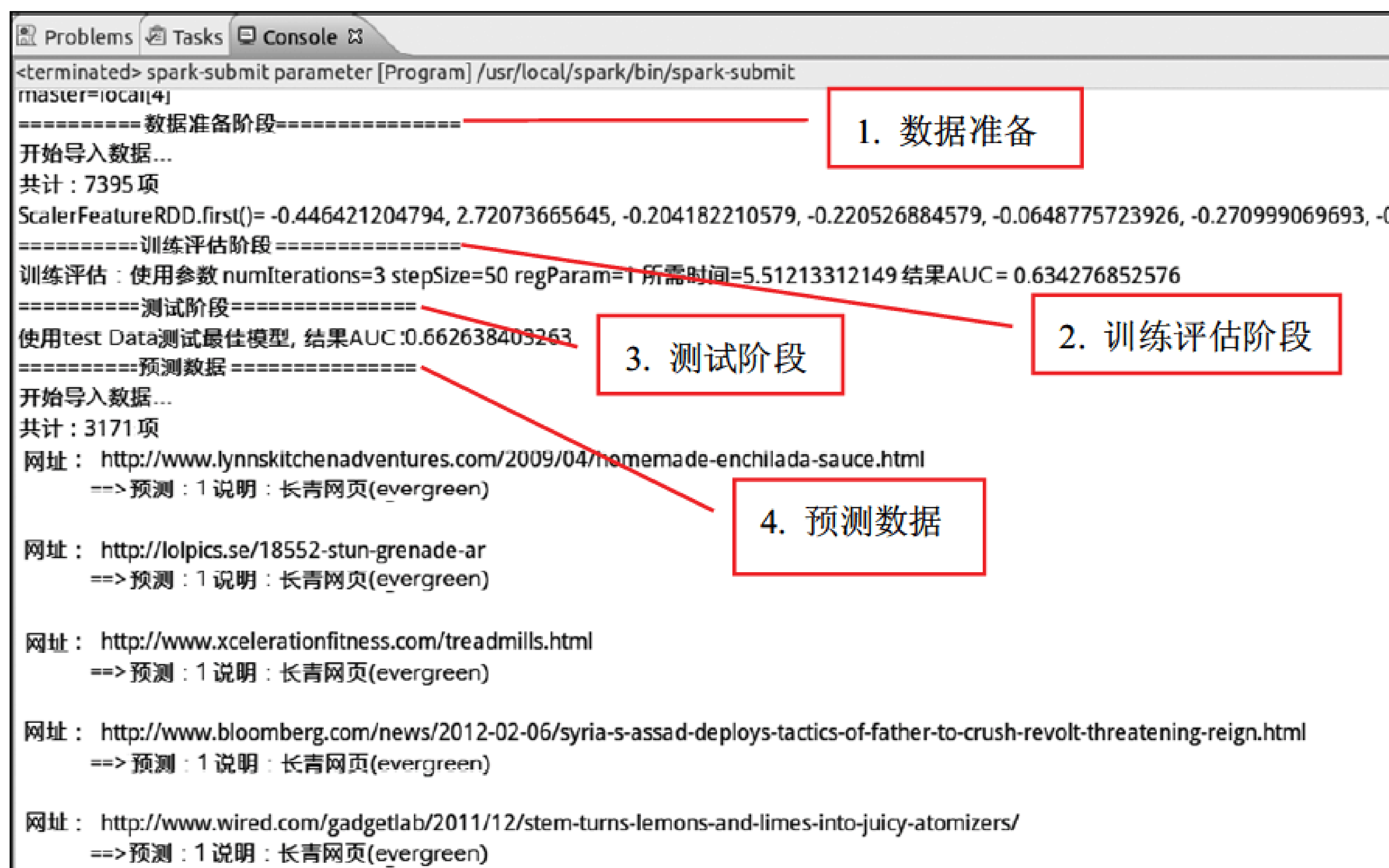


图 15-17　运行结果

（1）**数据准备阶段：**显示导入项数以及 trainData、validationData、testData 项数。

（2）**训练评估阶段：**运行最佳参数组合 numIterations=3、stepSize=50、miniBatchFraction = 1，结果 AUC = 0.63。

（3）**测试阶段：**使用测试数据再测试模型，AUC 大约为 0.66。与训练阶段差异不大，确认没有 overfitting 的问题。

（4）**预测阶段：**使用 test.tsv 预测是长青网页（evergreen）还是暂时性网页（ephemeral）。预测的数据会显示出网址，可以用浏览器查看一下是否准确。

15.5 结论

本章介绍了 Spark MLlib 支持向量机 SVM 的基本原理，完成了数据准备、训练模型、预测网页是暂时性还是长青，并调校参数找出最佳参数组合，提高预测准确度。下一章我们将介绍朴素贝叶斯二元分类。

第 16 章　Python Spark MLlib朴素贝叶斯二元分类

朴素贝叶斯（Naïve-Bayes）分析是简单而且实用的分类方法。朴素贝叶斯分类法以贝氏定理（Bayes' theorem）为基础。贝氏定理来自于 18 世纪数学家托马斯·贝叶斯。朴素贝叶斯分类法希望能通过概率统计的分析来判断未知类的数据应该属于哪一类。

本章将使用第 13 章介绍的 StumbleUpon 数据集，运用朴素贝叶斯（Naïve-Bayes）二元分类来预测网页是暂时性（ephemeral）的还是长青的（evergreen），并通过训练评估找出最佳参数组合，提供预测准确度。

16.1 朴素贝叶斯分析原理的介绍

步骤01 朴素贝叶斯分析实例

朴素贝叶斯分析借助分析数据中的特征与标签之间的概率来作为分类的依据。

假设我们有 10 条训练样本如下：

项目（特征）	湿度（特征）	气压（特征）	风向（特征）	是否会下雨（标签）
1	<50	高	西	不会
2	51~60	低	东	不会
3	51~60	高	东	不会
4	>70	低	西	会
5	61~70	高	西	不会
6	61~70	中	西	会
7	51~60	高	南	会
8	51~60	中	南	会
9	61~70	低	南	会
10	<50	高	北	会

我们想要判断新进样本“高气压、湿度 51~60、西风，是否会下雨”的时候：

（1）首先计算“高气压、湿度 51~60、西风，会下雨的概率”。

（2）接下来计算“高气压、湿度 51~60、西风，不会下雨的概率”。

（3）如果“会下雨的概率”>“不会下雨的概率”，朴素贝叶斯分类预测新进样本“会下雨”。

（4）如果“会下雨的概率”<“不会下雨的概率”，朴素贝叶斯分类预测新进样本“不会下雨”。

步骤02 计算“高气压、湿度 51~60、西风，会下雨的概率”

首先，计算“高气压、湿度 51~60、西风，会下雨的概率”，方式如下：

$P(\text{会}) * P(\text{高} \mid \text{会}) * P(51\sim60 \mid \text{会}) * P(\text{西} \mid \text{会}) = (6/10) * (2/6) * (2/6) * (2/6) = 0.02222$

说明如下：

概率条件	计算方式	计算结果
P(会)	(会下雨的个数) / (全部个数)	(6/10)
P(高｜会)	(高气压 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(51~60｜会)	(湿度 51~60 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(西｜会)	(西风 且 会下雨的个数) / (会下雨的个数)	(2/6)
P(会)* P(高|会) * P(51~60 |会) *P(西|会)	(6/10)* (2/6) * (2/6) * (2/6)	0.02222

步骤03 计算“高气压、湿度 51~60、西风，不会下雨的概率”

接下来，计算“高气压、湿度 51~60、西风，不会下雨的概率”，方式如下：

P(不会) * P(高|不会)*P(51~60|不会)*P(西|不会)=(4/10)*(3/4)*(2/4) *(2/4)= 0.075

说明如下：

概率条件	计算方式	计算结果
P(不会)	(不会下雨的个数) / (全部个数)	(4/10)
P(高｜不会)	(高气压 且 不会下雨的个数) / (不会下雨的个数)	(3/4)
P(51~60｜不会)	(湿度 51~60 且 不会下雨的个数) / (不会下雨的个数)	(2/4)
P(西｜不会)	(西风 且 不会下雨的个数) / (不会下雨的个数)	(2/4)
P(不会) * P(高|不会)* P(51~60|不会)* P(西|不会)	(4/10)*(3/4)*(2/4) *(2/4)	0.075

结论：“会下雨”的概率为 0.02222，小于“不会下雨”的概率 0.075，所以朴素贝叶斯分类预测新进样本“不会下雨”。

16.2 RunNaiveBayesBinary.py 程序说明

我们已经使用 Spark MLlib 创建逻辑回归分析 RunNaiveBayesBinary.py，完整的程序代码可参考本书提供的范例程序。RunNaiveBayesBinary.py 程序的架构与我们第 13 章介绍的 RunDecisionTreeBinary.py 类似，读者可以参考第 13 章的详细说明。下面仅说明 RunSVMWithSGDBinary.py 重要的修改部分。

步骤01 导入 NaiveBayes 模块（见图 16-1）

```
RunNaiveBayesBinary
# -*- coding: UTF-8 -*-
import sys
from time import time
import pandas as pd
import matplotlib.pyplot as plt
from pyspark import SparkConf, SparkContext
from pyspark.mllib.classification import NaiveBayes
from pyspark.mllib.regression import LabeledPoint
import numpy as np
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.feature import StandardScaler
```

导入 NaiveBayes 模块

图 16-1 导入 NaiveBayes 模块

步骤02 修改 PrepareData() 加入数据标准化的程序代码

我们将把 StumbleUpon 数据集运用于朴素贝叶斯二元分类。当进行贝氏二元分类时，必须对数值特征（Numerical features）字段进行标准化。因为数值特征字段单位不同而数字差异很大，所以彼此无法比较。这时就要使用标准化，让数值特征字段有共同的标准。

修改 PrepareData()，如图 16-2 所示。

```
print " 标准化之后 ",
stdScaler = StandardScaler(withMean=False, withStd=True).fit(featureRDD)
ScalerFeatureRDD=stdScaler.transform(featureRDD)
for i in ScalerFeatureRDD.first():
    print (str(i)+","),
```

加入数据标准化 withMean=false

图 16-2 修改 PrepareData() 加入数据标准化的程序代码

上述程序与之前第 15.2 节的做法类似，自行参考第 15.2 节的说明。但是朴素贝叶斯二元分类法在进行数据标准化时，参数 withMean 必须设置为 false。

步骤03 修改 convert_float() 将负数转换为 0

因为 NaiveBayes 数值特征字段一定要大于 0，所以修改 convert_float()，加入下述命令将负数转换为 0（见图 16-3）。

```
def convert_float(x):
    ret=(0 if x=="?" else float(x))
    return(0 if ret<0 else ret)
```

NaiveBayes 数值特征字段一定要大于 0，所以负数转换为 0

图 16-3 将负数转换为 0

步骤04 trainModel 训练模型

接下来要使用 NaiveBayes.train 进行朴素贝叶斯分类法训练，可参考图 16-4 进行操作。

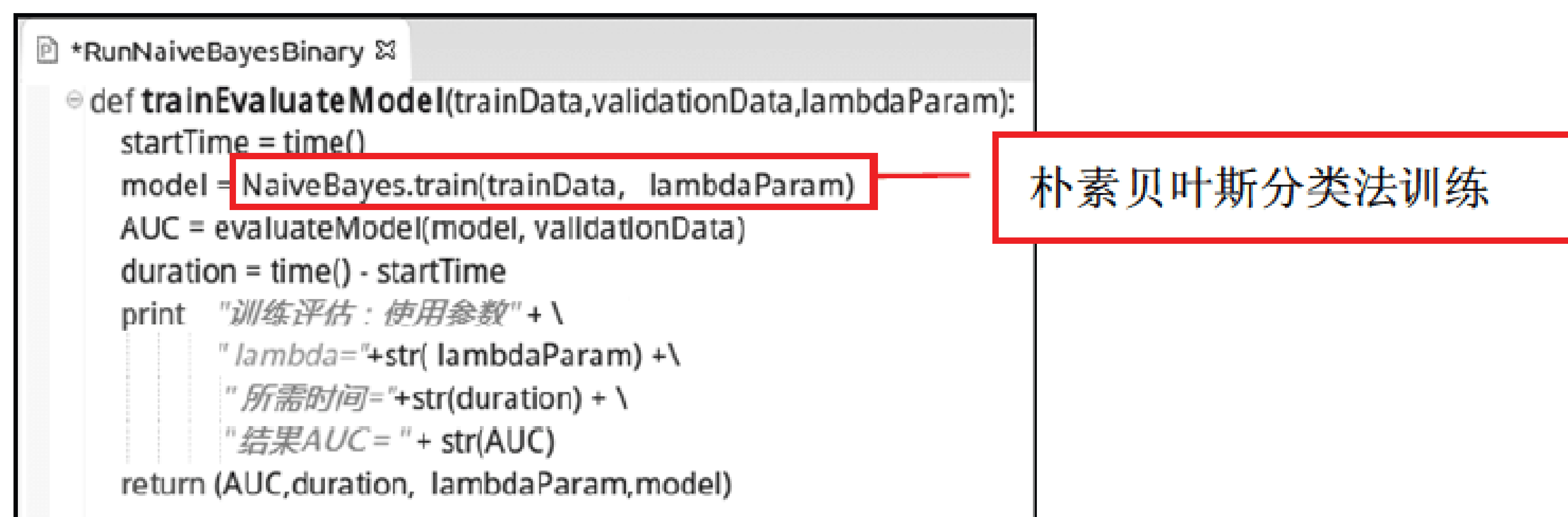


图 16-4　朴素贝叶斯分类法训练

在 NaiveBayes 必须输入下列参数：

NaiveBayesModel = NaiveBayes.train(input, lambda)	
返回	NaiveBayesModel
参数	说明
input	输入的训练数据 LabelPointRDD
lambda	设置 lambda 参数，默认值为 1.0

步骤 05 parametersEval 函数

输入 parametersEval 函数，如图 16-5 所示。



图 16-5　输入 parametersEval 函数

在这里评估的参数是 lambda。我们希望得知不同的参数值对准确率的影响以及运行所需要的时间。

16.3 运行 NaiveBayes.py 进行参数评估

步骤 01 运行外部工具（见图 16-6）

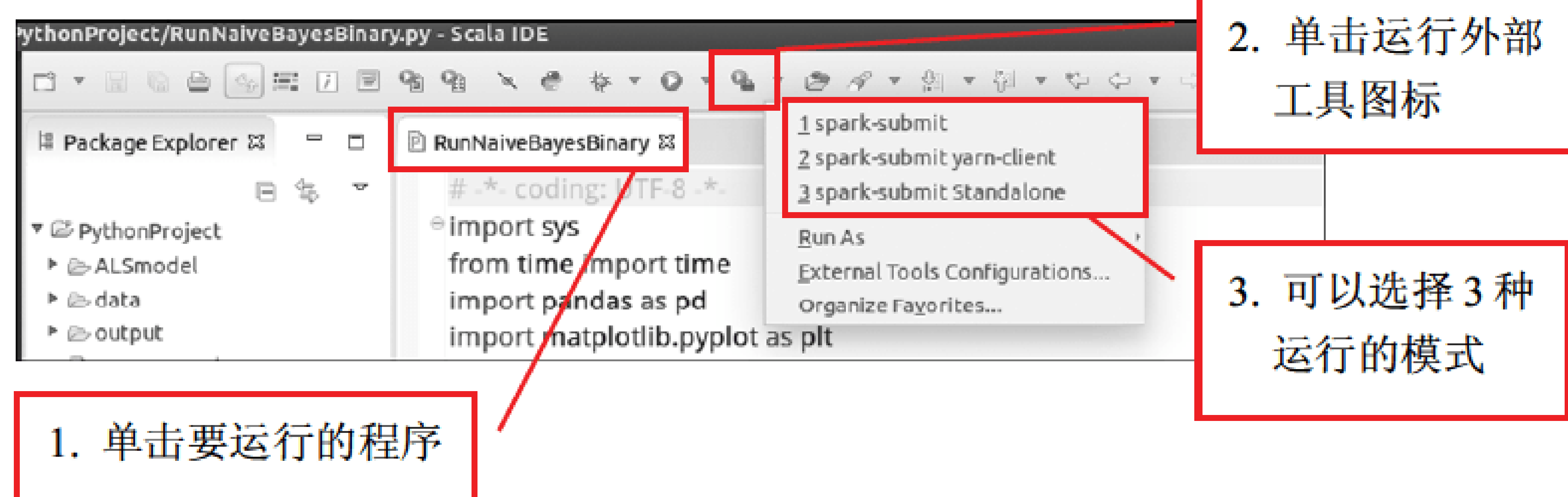


图 16-6　运行外部工具

步骤 02　输入参数“-e”（见图 16-7）



图 16-7　输入参数“-e”

步骤 03　lambda 参数评估

从图 16-8 可以看到 lambda = 60 时 AUC 最大，不过差异不大。lambda 越大，所需时间越少。

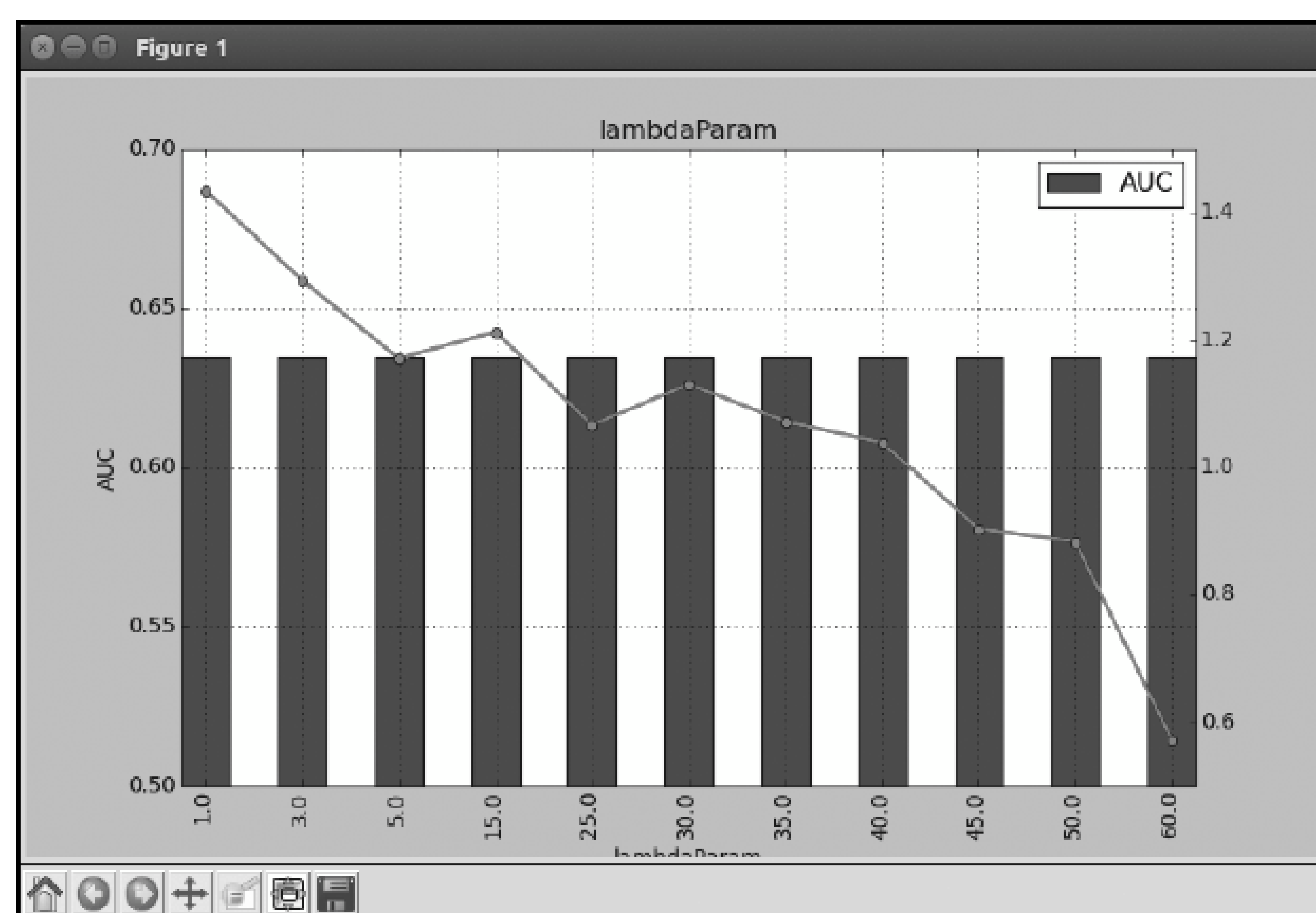


图 16-8　lambda 参数评估

16.4　运行训练评估并找出最好的参数组合

接下来，我们以 evaluateAllParameter 函数训练评估 3 种参数，希望找出最好的参数组合。

步骤 01　运行外部工具（见图 16-9）

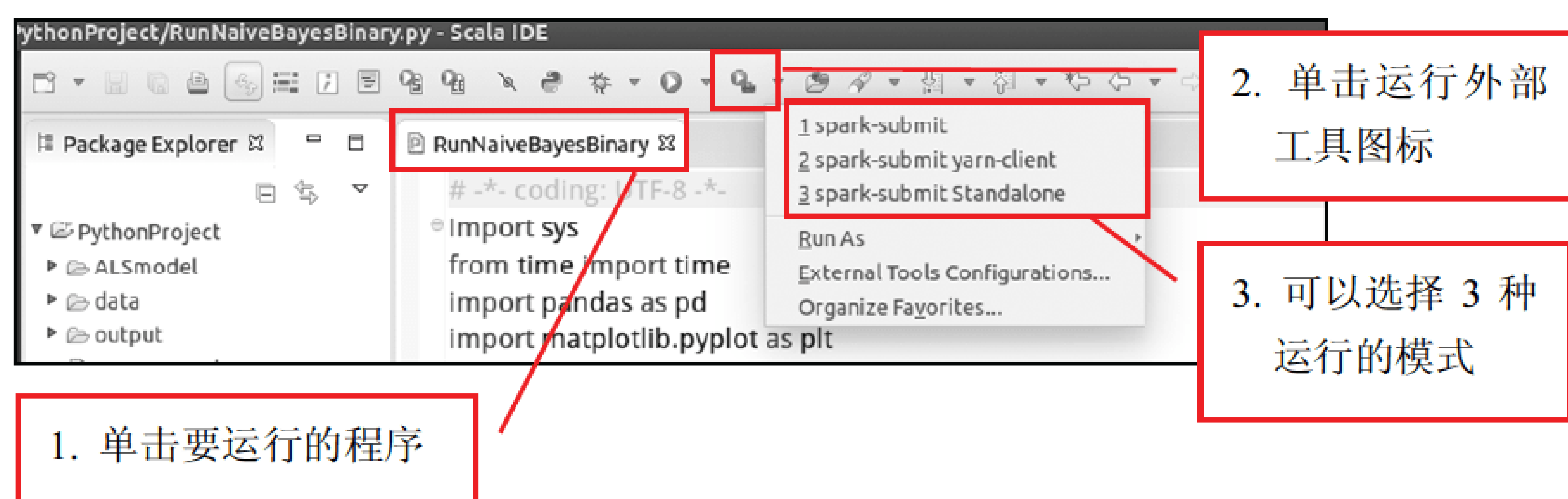


图 16-9　运行外部工具

步骤02 输入参数“-a”（见图 16-10）

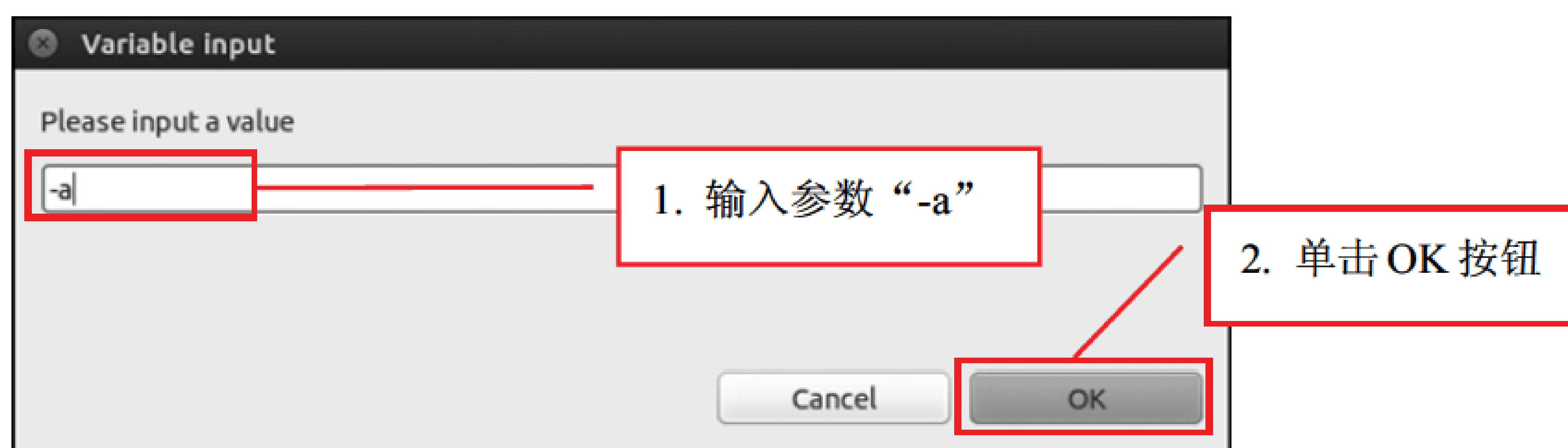


图 16-10　输入参数“-a”

步骤03 进行训练评估并找出最佳参数组合（见图 16-11）

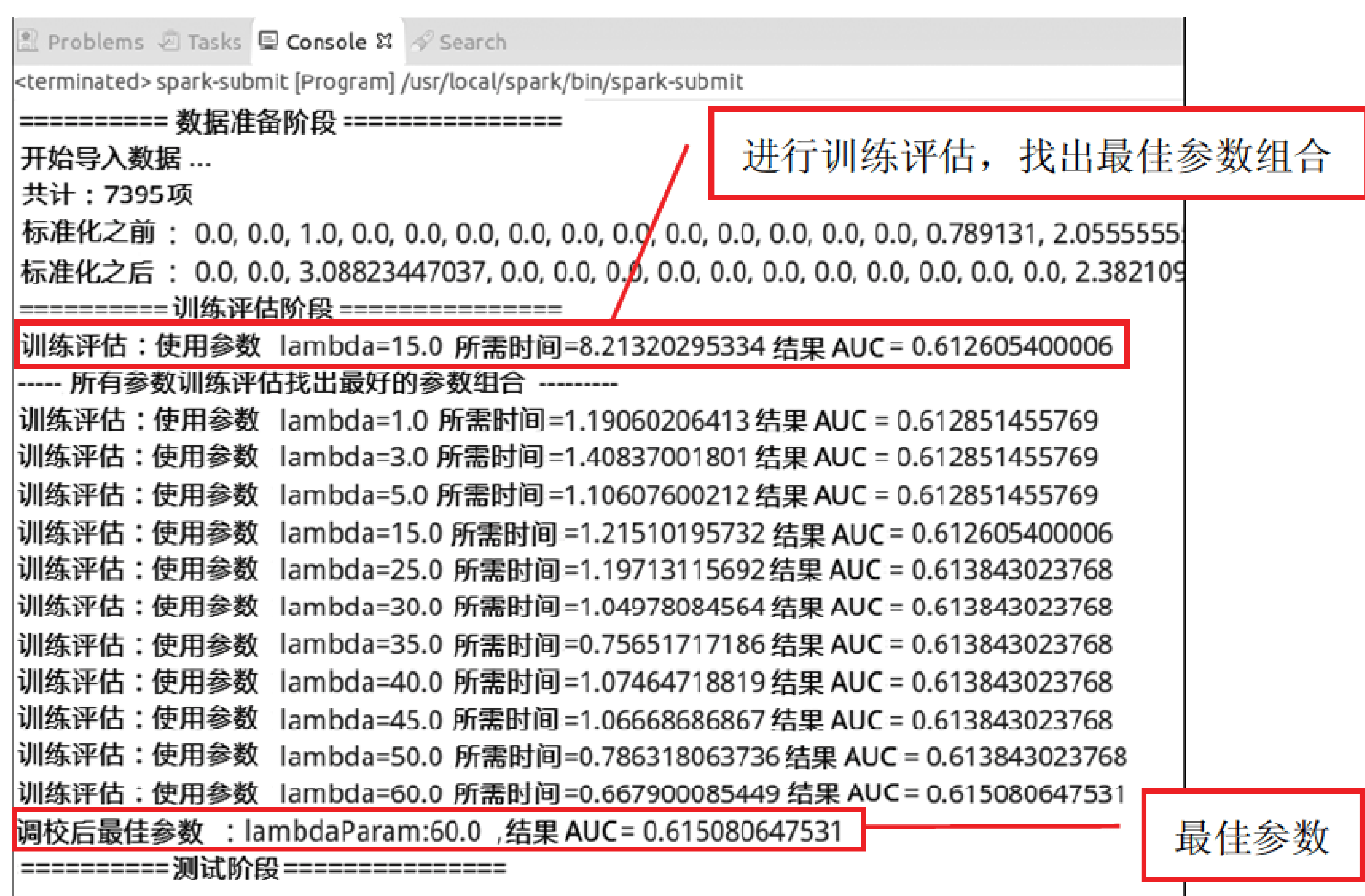


图 16-11　找出最佳参数组合

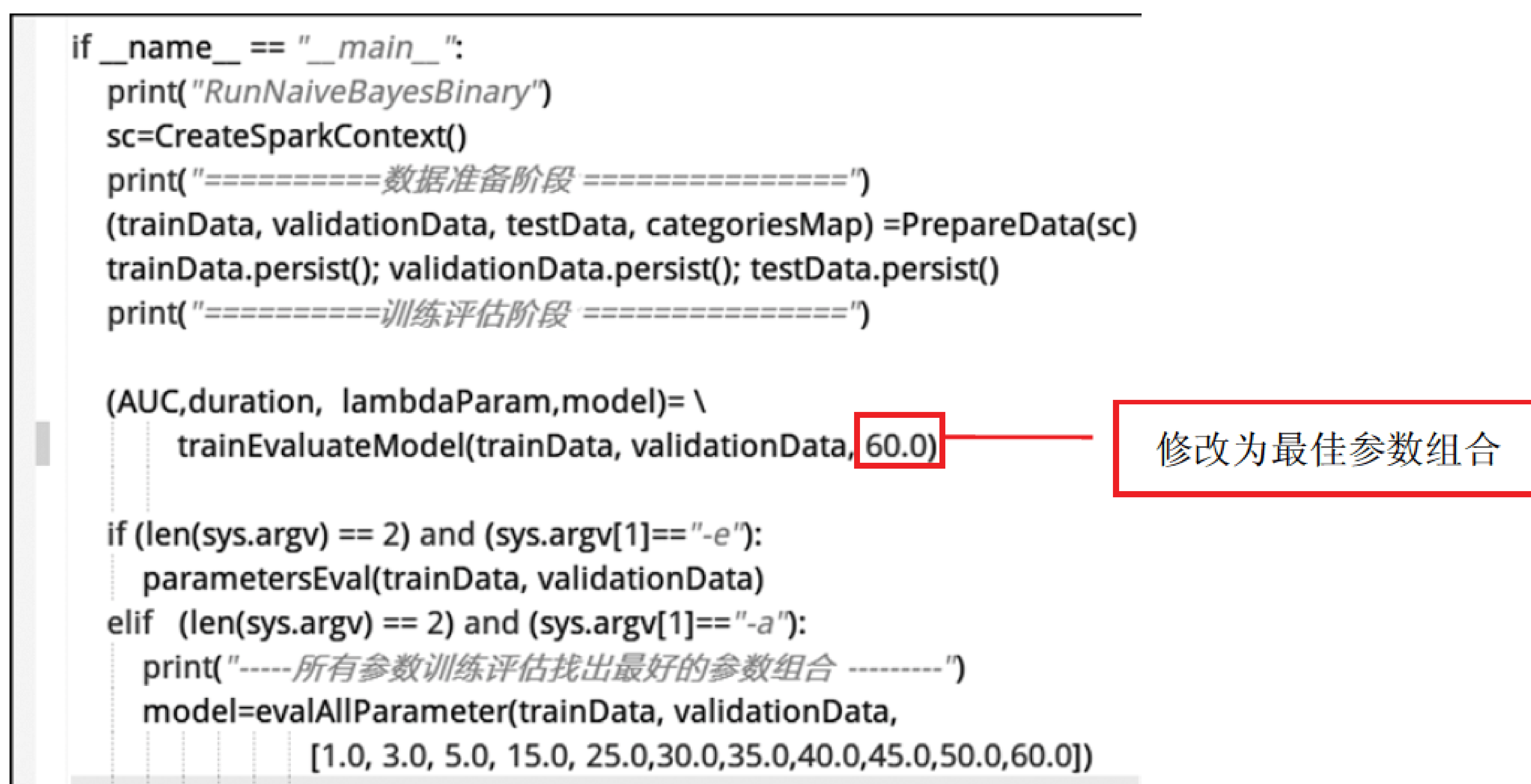
16.5 修改 RunNaiveBayesBinary.py 直接使用最佳参数进行预测

因为训练评估比较费时，所以当我们已经找出最佳参数组合就可以修改 trainEvaluateModel 程序为最佳参数组合。下次要进行预测时，可以不再运行训练评估，直接使用最佳参数进行预测即可。

步骤01 修改 main 为最佳参数组合

可以修改 main 程序，改为训练评估得到的最佳参数组合，如图 16-12 所示。





```
if __name__ == "__main__":
    print("RunNaiveBayesBinary")
    sc=CreateSparkContext()
    print("==========数据准备阶段 ===============")
    (trainData, validationData, testData, categoriesMap) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("==========训练评估阶段 ===============")

    (AUC,duration,  lambdaParam,model)= \
          trainEvaluateModel(trainData, validationData, 60.0)

    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
    elif   (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
        print("-----所有参数训练评估找出最好的参数组合 ---------")
        model=evalAllParameter(trainData, validationData,
                  [1.0, 3.0, 5.0, 15.0, 25.0,30.0,35.0,40.0,45.0,50.0,60.0])
```


图 16-12　修改 main 为最佳参数组合

步骤 02 开始运行程序（见图 16-13）

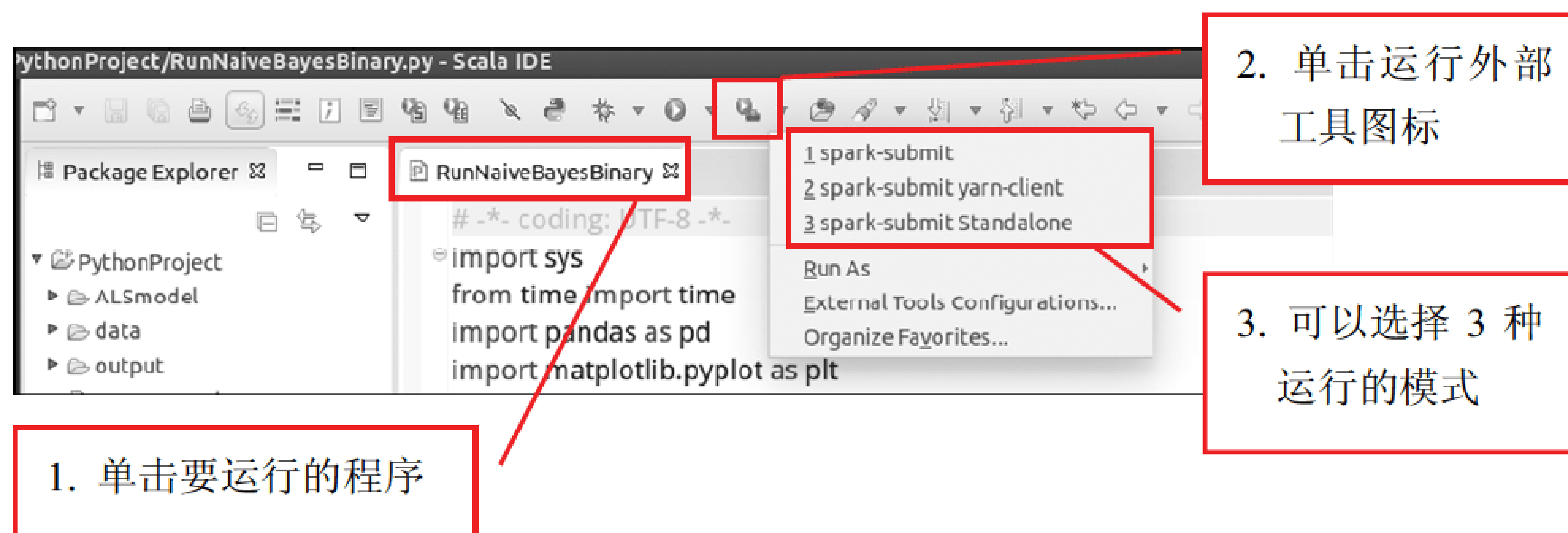


图 16-13　开始运行程序

步骤 03 不输入参数（见图 16-14）



图 16-14　不输入参数

步骤 04 运行结果

运行结果如图 16-15 所示。可以看到程序训练完成后就可以进行预测。因为不需要进行训练评估，所以运行所需的时间比较少。

```
Problems  Tasks  Console  Search
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit
RunNaiveBayesBinary
16/08/17 01:36:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfor
master=local[*]
========== 数据准备阶段 ===============
开始导入数据 ...
共计：7395 项
标准化之前 ： 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.789131, 2.055555556, 0
标准化之后 ： 0.0, 0.0, 3.08823447037, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.38210990 5
==========训练评估阶段 ===============
训练评估：使用参数  lambda=60.0 所需时间=6.62339711189 结果AUC = 0.650077399381
==========测试阶段 ===============
使用test Data测试最佳模型，结果 AUC:0.613032508085
==========预测数据 ===============
开始导入数据 ...
共计：3171 项
 网址：  http://www.lynnskitchenadventures.com/2009/04/homemade-enchilada-sauce.html
        ==>预测:1.0 说明:长青网页(evergreen)

 网址：  http://lolpics.se/18552-stun-grenade-ar
        ==>预测:1.0 说明:长青网页(evergreen)
```

1. 数据准备
2. 训练评估阶段
3. 测试阶段
4. 预测数据

图 16-15　运行结果

（1）**数据准备阶段：** 显示导入项数以及 trainData、validationData、testData 项数。

（2）**训练评估阶段：** 找出最佳参数组合 lambda = 60、AUC = 0.65。

（3）**测试阶段：** 使用测试数据再测试模型，AUC 大约为 0.61。与训练阶段差异不大，确认没有 overfitting 的问题。

（4）**预测阶段：** 使用 test.tsv 预测是长青网页（evergreen）还是暂时性网页（ephemeral）。预测的数据会显示出网址，可以用浏览器查看一下是否准确。

16.6 结论

本章介绍了 Spark MLlib 朴素贝叶斯二元分类的基本原理，完成了数据准备、训练模型、预测网页是暂时性还是长青，并调校参数找出最佳参数组合，提高预测准确度。下一章我们将介绍决策树多元分类。

第 17 章 Python Spark MLlib 决策树多元分类

本章将介绍决策树多元分类（DecisionTree Multi Class Classification），预测在不同条件下（Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等）适合种植的植被，并通过训练评估找出最佳参数组合，提高预测准确度。

17.1 “森林覆盖植被”大数据问题分析场景

森林的管理单位希望能运用大数据分析，帮助他们可以更节省人力、经费来管理森林，以提高森林覆盖率。因此找来了大数据分析师，与森林管理专员组成一个团队，负责“森林覆盖植被”大数据项目。

➢ **找出问题**

“问对问题”是解决问题的第一步。大数据分析师与森林管理专员讨论后发现，在一处森林中有各种树种，例如热带、针叶林等，每一块土地都有它适合生长的植被。如果能够预测哪些土地适合生长哪些植被，就能在适合的土地种植适当的植被，这样就可以节省人力、经费而达到事半功倍的效果，森林也可以长得更好。

➢ **设计解决方案模型**

大数据分析师与森林管理专员讨论哪些因素可能会影响适合生长的植被，经过讨论结果认为：Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等都会影响适合生长的植被。因此，大数据分析师整理数据如表 17-1 所示。

表 17-1　数据字段说明

字段	说明
feature 特征	Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等
label 标签 （预测目标）	Cover Type 森林覆盖分类： • 1——Spruce/Fir • 2——Lodgepole Pine • 3——Ponderosa Pine • 4——Cottonwood/Willow • 5——Aspen • 6——Douglas-fir • 7——Krummholz

➢ **搜集数据**

这些数据的搜集可能需要花费很多时间和人力，不过通常森林管理单位已经有长期田野调查的数据，并且借助现代科技，例如无人机空拍、卫星图等，可以搜集这些数据。

➢ **分析数据**

大数据分析师决定使用决策树多元分类（DecisionTree Multi-Class Classification）来创建

模型、训练、评估、预测数据。本章将详细介绍如何进行决策树多元分类分析。

➢ **其他类似的分类应用**

大家可能会觉得这个主题“森林覆盖植被”除了森林管理的专业外，其他日常生活可能应用不到。事实上这种类似的应用也很广泛，例如可以研究什么样的地点适合开设什么类型的店面，我们可以搜集相关数据。

- feature 特征：距离快捷公交站的距离、距离大学的距离、距离中学的距离、该地平均年收入、马路宽度、附近人口数、门口每小时人流量……
- label 标签（预测目标）：配合实际调查该地店面的类型，已经 3 年以上并且赚钱的店面类型（代表该地适合这种类型的店面）。

有了这些数据就可以创建模型、训练、评估，预测该地适合开设的店面类型。

17.2 UCI Covertype 数据集介绍

步骤 01 Machine Learning Repository 数据库

Machine Learning Repository 数据库是加州大学尔湾分校（University of California Irvine）提供的用于研究机器学习的数据库。数据库数量与种类繁多，还在不断增加。它的网址为 http://archive.ics.uci.edu/ml/。

步骤 02 UCI Covertype 数据集

其中有 Covertype 数据集，网址为 https://archive.ics.uci.edu/ml/datasets/Covertype。

Covertype Data Set 森林覆盖植被数据集是在一处森林中有各种树种植被，例如热带、针叶林等。图 17-1 为该数据集在网页上的下载链接。

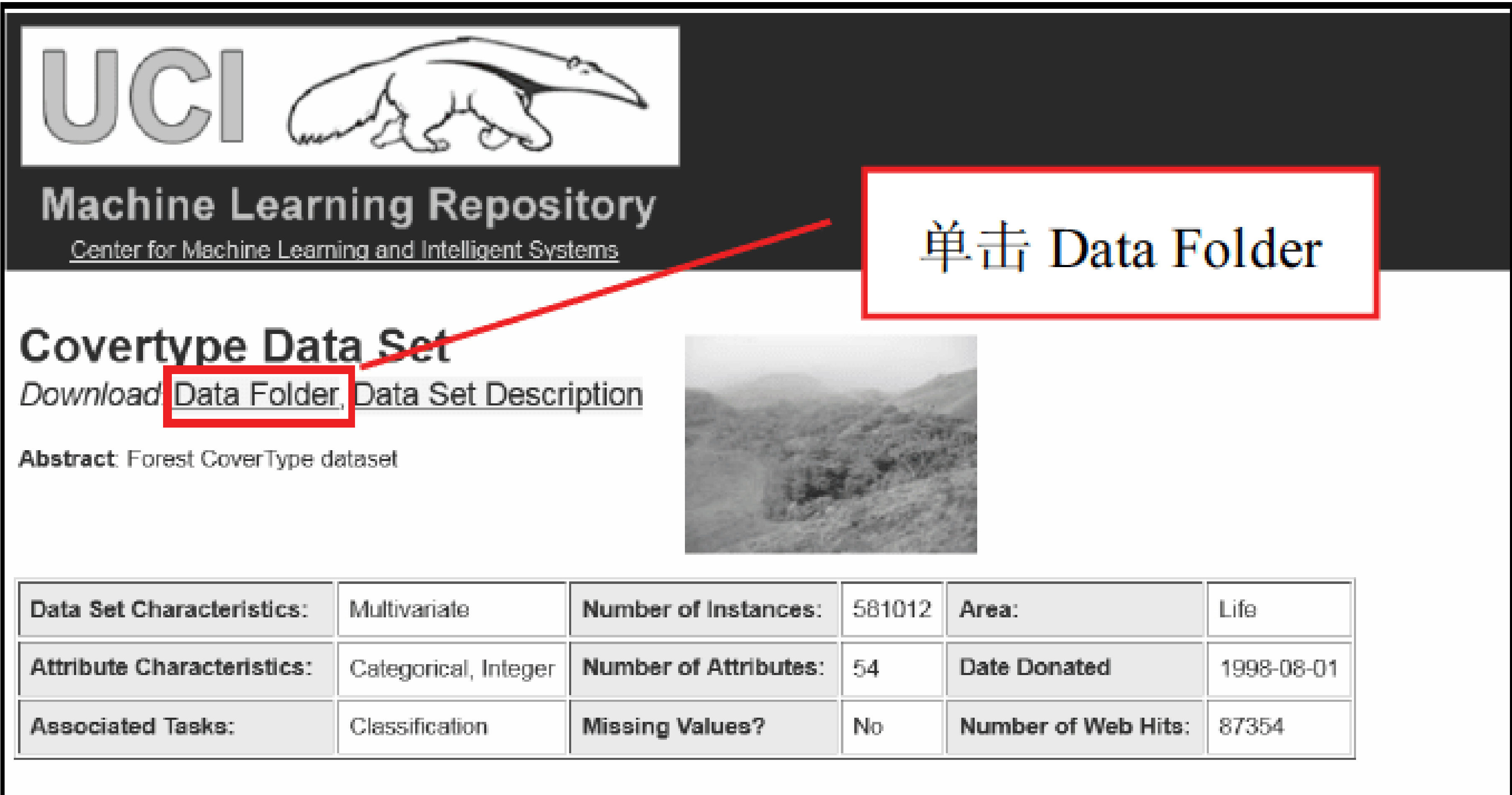


图 17-1　VCI Covertype 数据集

步骤03 查看 Data Folder（见图 17-2）

Index of /ml/machine-learning-databases/covtype

Name	Last modified	Size	Description
Parent Directory		-	
covtype.data.gz	31-Aug-1998 14:01	11M	
covtype.info	18-Apr-2010 00:01	14K	
old_covtype.info	31-Aug-1998 14:01	4.7K	

下载这个文件

Apache/2.2.15 (CentOS) Server at archive.ics.uci.edu Port 80

图 17-2　查看 Data Folder

17.3 下载与查看数据

步骤01 下载/解压缩文件

在“终端”程序中输入下列命令：

- **切换到项目数据目录**

```
cd ~/workspace/Classification/data
```

- **下载 covtype.data.gz**

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz
```

- **解压缩到当前目录**

```
gzip -d covtype.data.gz
```

运行后屏幕显示界面如图 17-3 所示。

```
hduser@master: ~/workspace/Classification/data
hduser@master:~$ cd ~/workspace/Classification/data
hduser@master:~/workspace/Classification/data$ wget https://archive.ics.uci.edu/
ml/machine-learning-databases/covtype/covtype.data.gz
--2016-05-18 14:47:34--  https://archive.ics.uci.edu/ml/machine-learning-databas
es/covtype/covtype.data.gz
正在解析主机 archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
正在连接 archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.249|:443... 已连接
。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 11240707 (11M) [application/x-gzip]
正在保存至: “covtype.data.gz”

100%[======================================>] 11,240,707  22.1KB/s   用时 5m 52s

2016-05-18 14:53:30 (31.2 KB/s) - 已保存 “covtype.data.gz” [11240707/11240707])

hduser@master:~/workspace/Classification/data$ gzip -d covtype.data.gz
```

图 17-3　下载/解压缩文件

步骤 02　设置“终端”程序的显示宽度

因为 covtype.data 字段很多，“终端”程序默认的字段宽度为 80×24，查看 covtype.data 时会换行，不容易阅读，所以将“终端”程序设置为 132×43。设置步骤如图 17-4 所示。

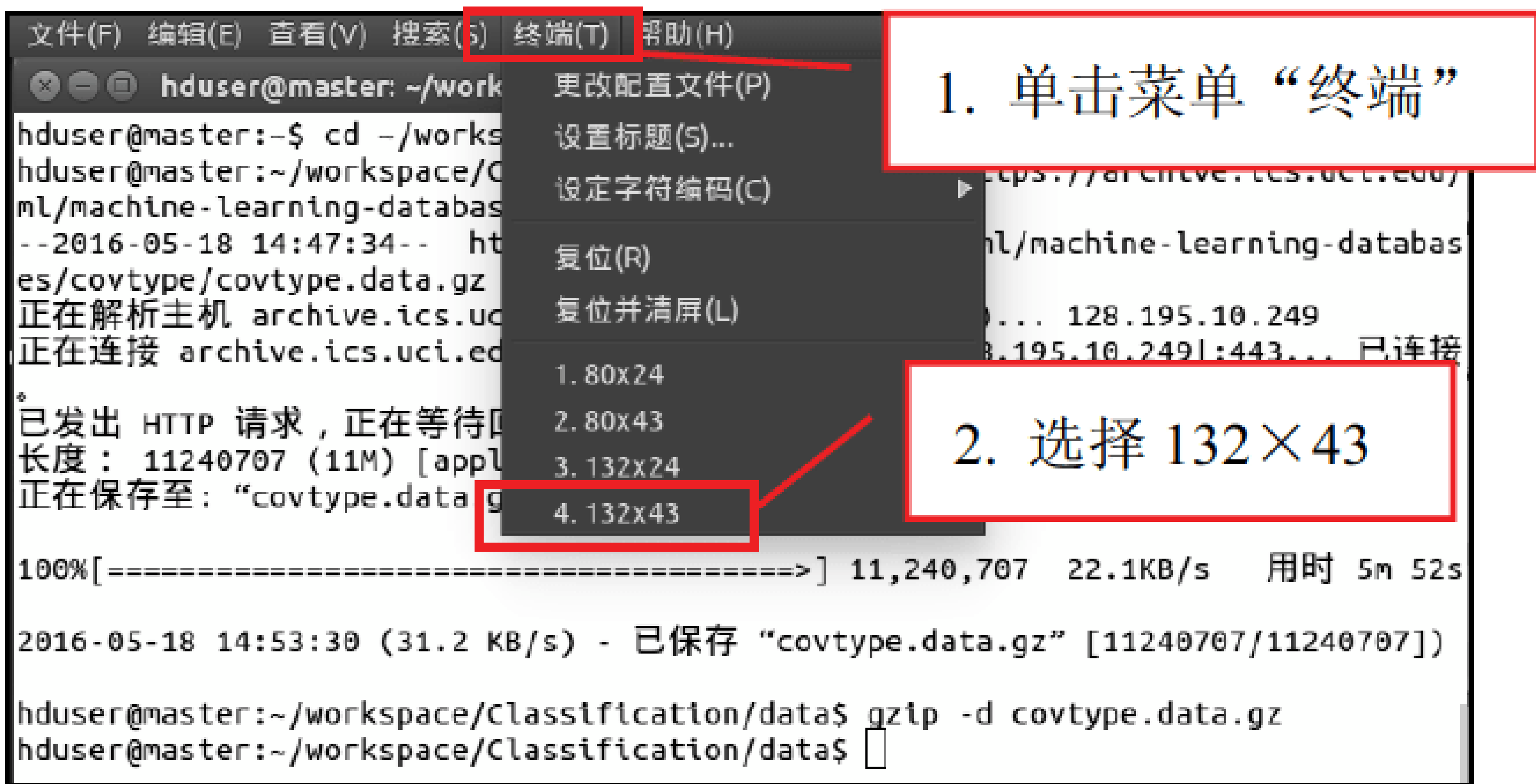


图 17-4　设置“终端”程序的显示宽度

步骤 03　查看 covtype.data 数据

使用下述命令查看 covtype.data：

```
cat covtype.data|more
```

运行后屏幕显示界面如图 17-5 所示。



图 17-5　查看 covtype data 数据

我们可以看到最后一个字段是 label 标签字段，其余是 feature 特征字段。各字段说明如表 17-2 所示。

表 17-2　covtype.data 中的字段

字段	字段分类	说明
字段 1~10	numerical features 数值特征字段	数值字段：例如 Elevation（海拔）、Aspect（方位）、Slope（斜率）、Vertical_Distance_To_Hydrology（距离水源的垂直距离）、Horizontal_Distance_To_Roadways（距离水源的水平距离）、Hillshade_9am（9 点时阴影）等

（续表）

字段	字段分类	说明
字段 11~14	categorical features 分类特征字段	Wilderness Areas 荒野分类字段： • 1——Rawah Wilderness Area • 2——Neota Wilderness Area • 3——Comanche Peak Wilderness Area • 4——Cache la Poudre Wilderness Area 已经编码成 OneHotEncoding。 例如：如果是 Neota Wilderness Area，就为 0,1,0,0；如果是 Comanche Peak Wilderness Area，就为 0,0,1,0
字段 15~54	categorical features 分类特征字段	Soil Types 土壤分类 已经编码成 OneHotEncoding
字段 55	label 标签字段（预测目标）	Cover Type 森林覆盖分类： • 1——Spruce/Fir • 2——Lodgepole Pine • 3——Ponderosa Pine • 4——Cottonwood/Willow • 5——Aspen • 6——Douglas-fir • 7——Krummholz

可以看到上述 categorical features 分类特征字段（荒野分类、土壤分类）都已经是编码成 OneHotEncoding 的数值字段，所以不需要自己转换为数值字段。

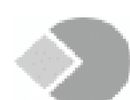
17.4 修改 PrepareData() 数据准备

决策树多元分析 RunDecisionTreeMulti.py 完整的程序代码请参考本书范例程序。决策树多元分类程序基本架构与第 13 章介绍的决策树二元分类 RunDecisionTreeBinary.py 类似，读者可以参考第 13 章的详细说明。本章仅说明 RunDecisionTreeMulti.py 重要的修改部分。

步骤 01 修改 PrepareData() 数据准备

PrepareData() 数据准备主要分为以下 3 个步骤：

```
def PrepareData(sc):
    #---------------1. 导入并转换数据 -------------
    #---------------2. 建立训练评估所需数据 RDD[LabeledPoint]-------------
    #---------------3. 以随机方式将数据分为 3 部分并返回 -------------
```

步骤02 导入并转换数据

输入下列程序代码：使用 sc.textFile 读取 covtype.data，然后 rawData 使用 map 与 lambda 语句传入 x 作为参数，调用 x.split(",")，以逗号“,”分隔字段，提取每一个字段。

```
print(" 开始导入数据 ...")
rawData = sc.textFile(Path+"data/covtype.data")
print(" 共计: " + str(rawData.count()) + " 笔 ")
lines = rawData.map(lambda x: x.split(","))
```

步骤03 建立训练评估所需数据 RDD[LabeledPoint]

建立 LabelPoint，由标签字段与特征字段组成：

- 标签字段：调用 extract_label(r) 获取标签字段，步骤 4 会介绍。
- 特征字段：extract_features(r,len(r) - 1)，获取除了最后字段的所有字段，步骤 5 会介绍。

步骤04 extract_label () 提取 label 字段

编写 extract_label ()程序来提取 label 字段：

```
def extract_label(record):
    label=(record[-1])
    return float(label)-1
```

以上程序代码说明如下：

（1）先使用 label=(record[-1])取出最后一个字段，就是 label 标签字段。
（2）再使用 return float(label)-1 返回 label 转换为 float 后再减 1 的结果。

为何 label 值要减 1 呢？这是因为要执行 DecisionTree.trainClassifier 训练，规定 label 一定是从 0 开始的数值。但是原本 label 的值范围是 1~7（1——Spruce/Fir、2——Lodgepole Pine、3——Ponderosa Pine、4——Cottonwood/Willow、5——Aspen、6——Douglas-fir、7——Krummholz），所以必须使用“float(label)-1”，使 label 值范围变成 0~6。

步骤05 extract_feature () 提取特征字段

以下程序代码 extract_feature() 用来提取特征字段：

```
def extract_features(record,featureEnd):
    numericalFeatures=[convert_float(field) for field in record[0:
    featureEnd]] return numericalFeatures
```

以上程序代码说明如下：

（1）因为我们是读取文本文件，数据类型是 string，所以每一个字段必须转换为 float。
（2）[convert_float(field) for field in record[0: featureEnd]]会从第 0 字段到 featureEnd 字

段，执行 convert_float 函数，转换为 float。

（3）以上 featureEnd 是参数，当我们调用 extract_features 时会传入 len(r)–1，也就是会读取到倒数第 2 个字段。

（4）最后将转换后的结果存入 numericalFeatures 并返回。

步骤 06 以随机方式将数据分为 3 个部分并返回

以下程序代码以 .randomSplit([8, 1, 1]) 按照 8:1:1 的比例以随机方式分为 3 个部分（trainData, validationData, testData）并返回数据。

```
#----------------3. 以随机方式将数据分为 3 个部分并且返回 -------------
(trainData, validationData, testData) = labelpointRDD.randomSplit([8, 1, 1])
print(" 将数据分 trainData:" + str(trainData.count())+\
        "    validationData:" + str(validationData.count()) +\
        "    testData:" + str(testData.count()))
print labelpointRDD.first()
return (trainData, validationData, testData)
```

17.5 修改 trainModel 训练模型程序

步骤 01 修改 trainModel 训练模型

在决策树多元分类中，我们的 label 标签字段有 7 种可能值，所以分类数目 numClasses 设置为 7，并且使用 accuracy 方式评估，如图 17-6 所示。

```
def trainEvaluateModel(trainData,validationData,
                       impurityParm, maxDepthParm, maxBinsParm):
    startTime = time()
    model = DecisionTree.trainClassifier(trainData,\
                     numClasses=7, categoricalFeaturesInfo={}, \
                     impurity=impurityParm,
                     maxDepth=maxDepthParm,
                     maxBins=maxBinsParm)
    accuracy = evaluateModel(model, validationData)
    duration = time() - startTime
    print    "训练评估：使用参数 " + \
         " impurityParm= %s"%impurityParm+ \
         " maxDepthParm= %s"%maxDepthParm+ \
         " maxBinsParm = %d."%maxBinsParm + \
          " 所需时间=%d"%duration + \
          " 结果accuracy = %f " % accuracy
    return (accuracy,duration, impurityParm, maxDepthParm, maxBinsParm,model)
```

numClasses 原本为 2 个修改为 7

改为使用 accuracy 方式评估

图 17-6 修改 trainModel 训练模型

步骤 02 修改 evaluateModel 评估模型

在多元分类（Multiclass Classification）评估模型中，我们使用 MulticlassMetrics 先建立 Metrics，然后使用 .accuracy 方法计算准确率，程序如图 17-7 所示。

```
RunDecisionTreeMulti
def evaluateModel(model, validationData):
    score = model.predict(validationData.map(lambda p: p.features))
    scoreAndLabels=score.zip(validationData.map(lambda p: p.label))
    metrics = MulticlassMetrics(scoreAndLabels)
    accuracy = metrics.accuracy
    return( accuracy)
```


图 17-7　修改 evaluateModel 评估模型

步骤 03 修改 evaluateParameter 绘图程序代码

在多元分类（Multiclass Classification ）中，我们使用 accuracy 评估模型的准确率，所以修改 evaluateParameter 绘图程序代码，可参考图 17-8 进行操作。

```
def evalParameter(trainData, validationData, evaparm,impurityList, maxDepthList, maxBinsList):
    metrics = [trainEvaluateModel(trainData, validationData,  impurity,numIter,  maxBins  )
               for impurity in impurityList for numIter in maxDepthList  for maxBins in maxBinsList ]
    if evaparm=="impurity":
        IndexList=impurityList[:]
    elif evaparm=="maxDepth":
        IndexList=maxDepthList[:]
    elif evaparm=="maxBins":
        IndexList=maxBinsList[:]
    df = pd.DataFrame(metrics,index=IndexList,
               columns=['accuracy', 'duration','impurity', 'maxDepth', 'maxBins','model'])

    showchart(df,evaparm,'accuracy','duration',0.6,1.0 )
```


图 17-8　修改 evaluateParameter 绘图程序代码

17.6 使用训练完成的模型预测数据

步骤 01 输入 PredictData 程序代码

PredictData() 进行预测主要分为以下 3 个步骤：

```
def PrepareData(sc):
    #---------------1. 导入并转换数据 -------------
    #---------------2. 建立预测所需数据 RDD[LabeledPoint]-------------
    #---------------3. 进行预测并显示结果 ------------
```

以上第 1、2 步与 PrepareData() 相同，可自行参考第 17.4 节，第 3 步进行预测并显示结果（将在步骤 2 进行介绍）。

步骤 02 进行预测并显示结果

进行预测并显示结果，程序代码如下：

```
for lp in labelpointRDD.take(100):
    predict = model.predict(lp.features)
    label=lp.label
    features=lp.features
    result = (" 正确 " if (label == predict) else " 错误 ")
    print(" 土地条件：海拔 :" + str(features[0]) +
            " 方位 :" + str(features[1]) +
            " 斜率 :" + str(features[2]) +
            " 水源垂直距离 :" + str(features[3]) +
            " 水源水平距离 :" + str(features[4]) +
            " 9 点时阴影 :" + str(features[5]) +
            "....==> 预测 :" + str(predict) +
            " 实际 :" + str(label) + " 结果 :" + result)
```

以上程序代码的详细说明如表 17-3 所示。

表 17-3　程序代码说明

程序代码	说明
for lp in labelpointRDD.take(20):	取出 20 项进行预测
predict = model.predict(lp.features)	使用训练完成的模型传入特征 lp.features 进行预测，预测结果存放于 predict 变量
label=lp.label	取得标签字段
features=lp.features	取得特征字段
result = (" 正确 " if (label == predict) else " 错误 ")	判断预测是否正确
print(" 土地条件：海拔 :" + …	打印特征字段、实际与结果



17.7 运行 RunDecisionTreeMulti.py 进行参数评估

步骤 01 运行外部工具（见图 17-9）

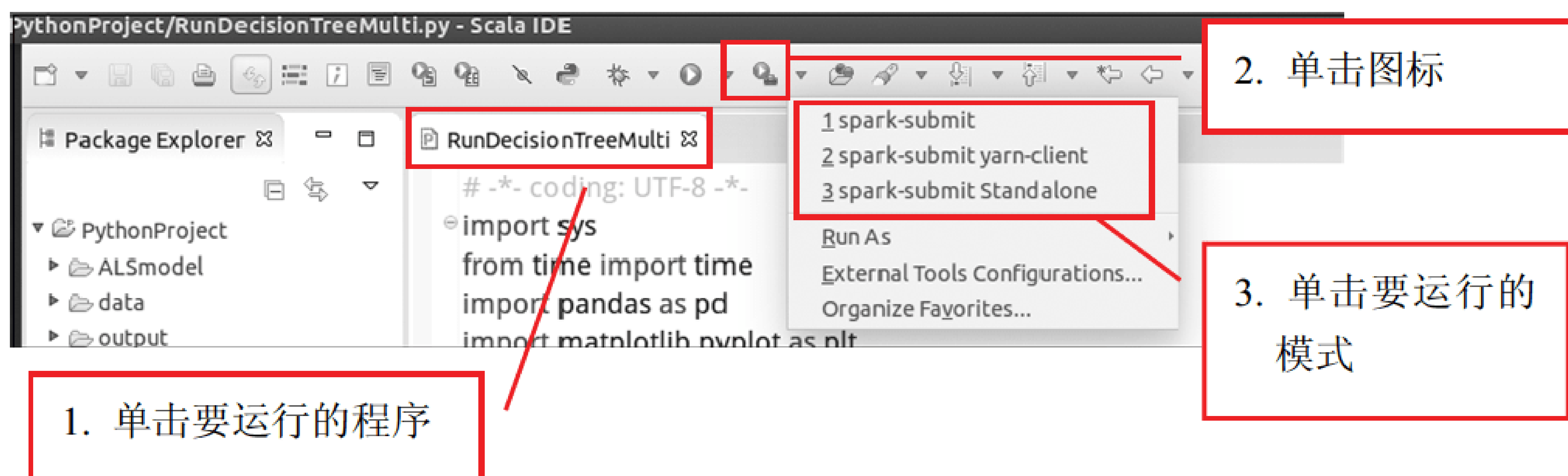


图 17-9　运行外部工具

步骤 02 输入参数 "-e"（见图 17-10）



图 17-10　输入参数 "-e"

步骤 03 impurity 参数评估

直方图代表 accuracy，折线图代表时间。从图 17-11 可以看到 gini 准确度比 entropy 好一些，但是并没有太大差别。不过，运行时间 gini 所需的时间是 entropy 的 4 倍。

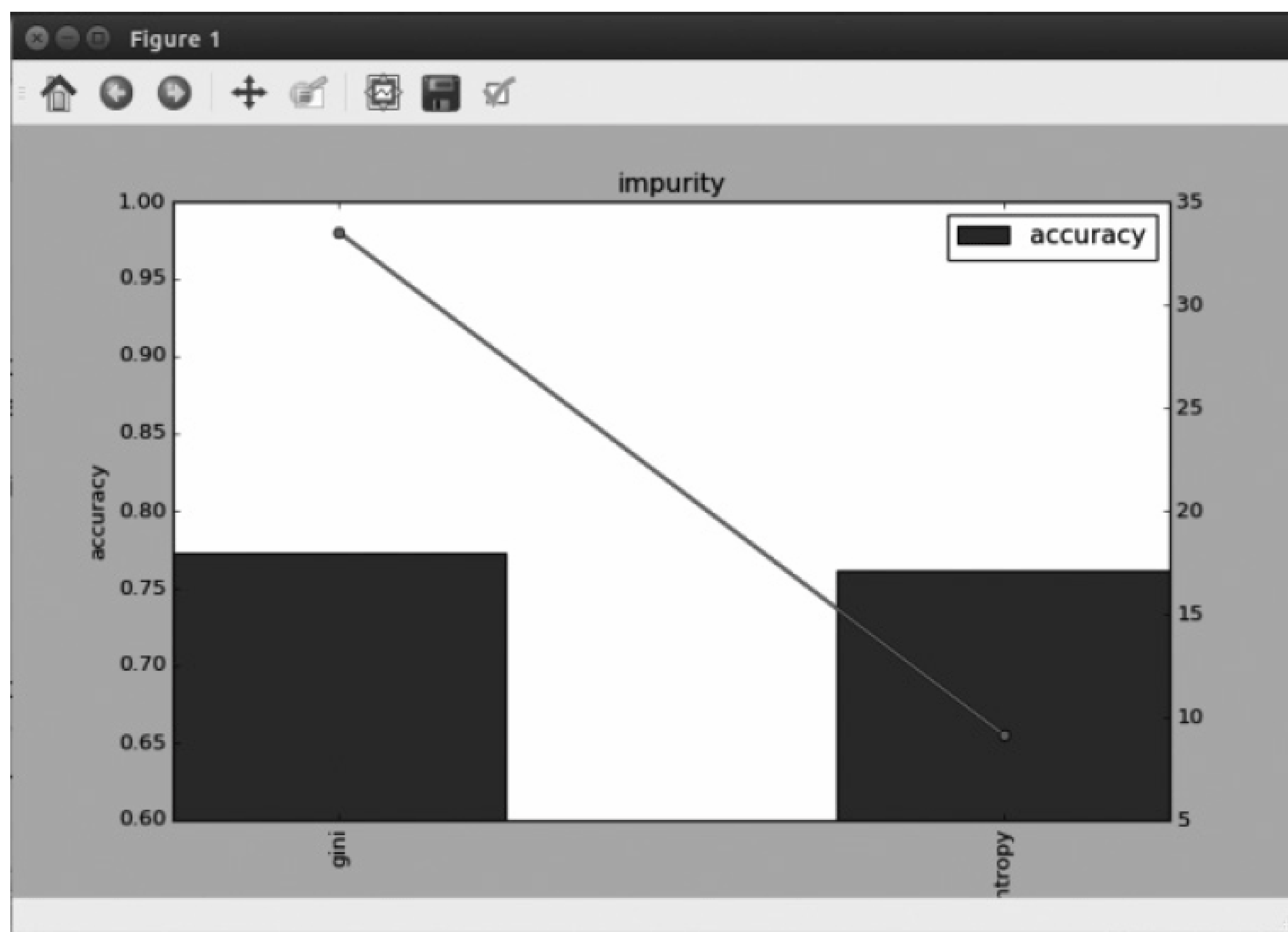


图 17-11　impurity 参数评估

步骤04 maxDepth 参数评估

从图 17-12 可以看到 maxDepth= 25 时 accuracy 最高；maxDepth 越大，所需的时间越多。

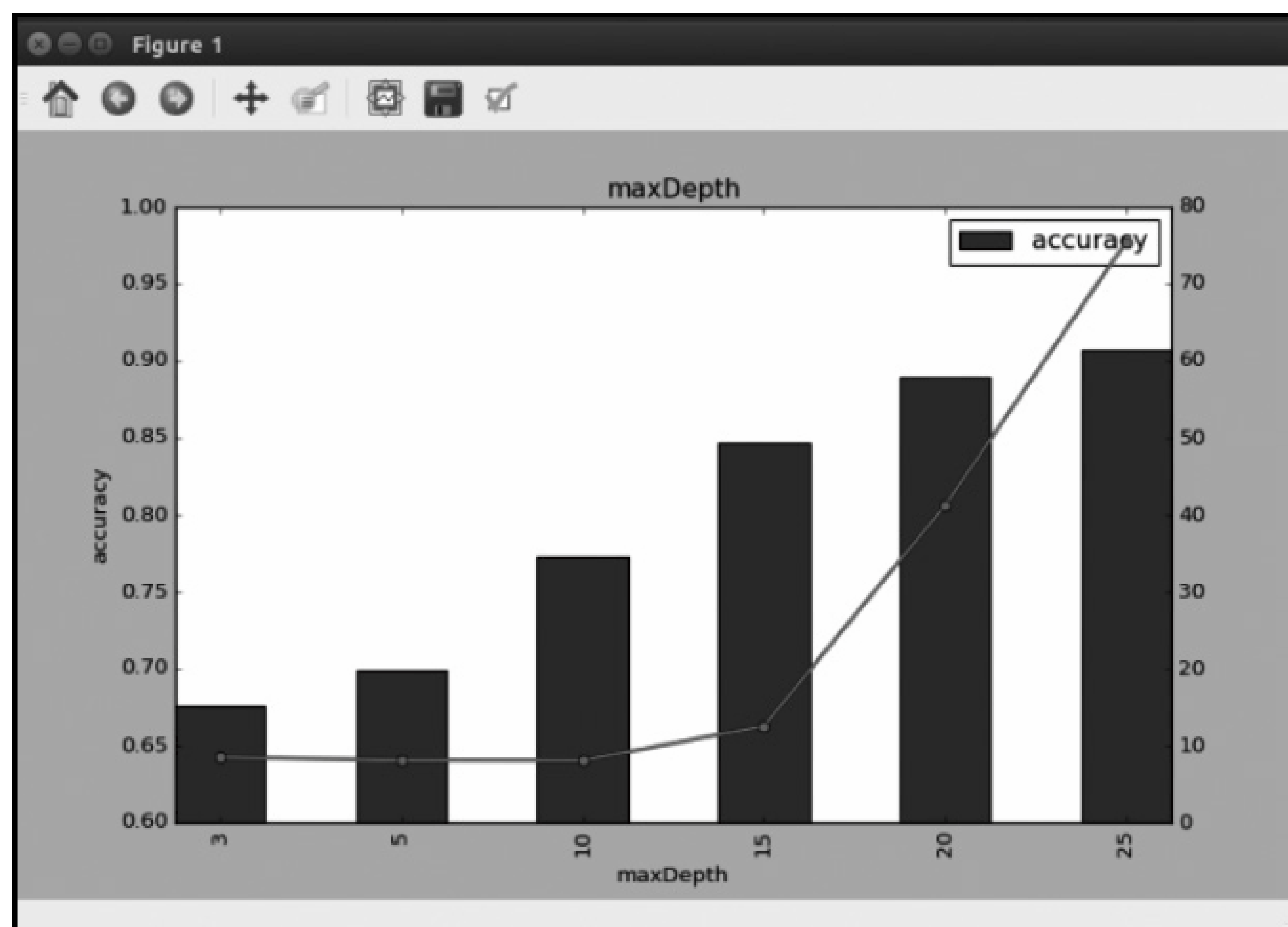


图 17-12　maxDepth 参数评估

步骤05 maxBins 参数评估

从图 17-13 可以看到 maxBins = 200 时 accuracy 最高；maxBins 越大，所需的时间越多。



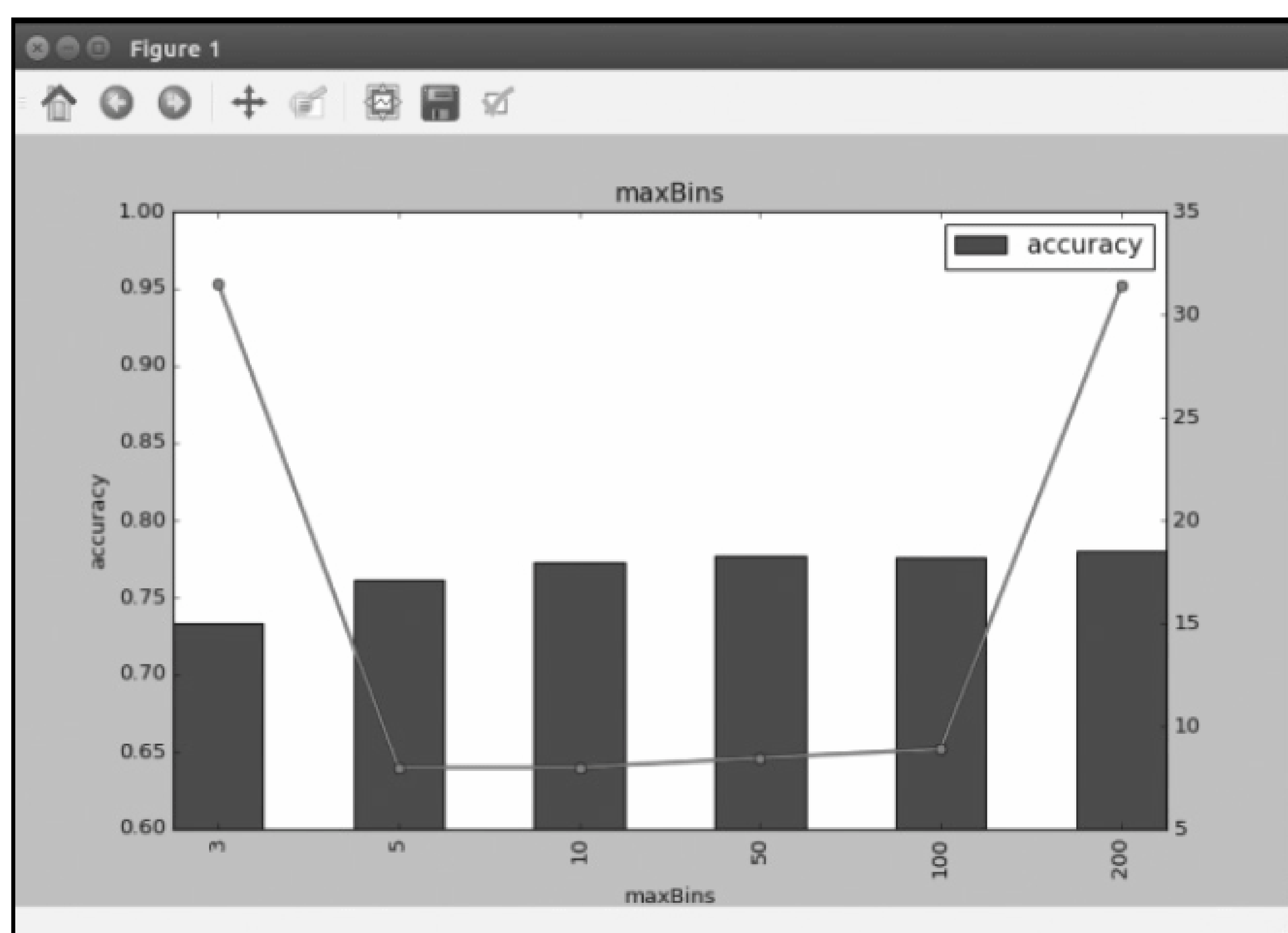


图 17-13　maxBins 参数评估

17.8 运行 RunDecisionTreeMulti.py 训练评估参数并找出最好的参数组合

接下来，我们以 evaluateAllParameter 函数训练评估 3 种参数，希望找出最好的参数组合。

步骤 01 输入参数“-a”（见图 17-14）



图 17-14　输入参数“-a”

步骤 02 进行训练评估找出最佳参数组合（见图 17-15）

```
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit
========== 数据准备阶段 ===============
开始导入数据 ...
共计：581012项
建立训练评估所需数据...
将数据分trainData:465031   validationData:58402   testData:57579
(4.0,[2596.0,51.0,3.0,258.0,0.0,510.0,221.0,232.0,148.0,6279.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,C
========== 训练评估阶段 ===============
训练评估：使用参数  impurityParm= entropy maxDepthParm= 25 maxBinsParm = 100. 所需时间=113 结果accuracy = 0.937297
----- 所有参数训练评估找出最好的参数组合 ---------
训练评估：使用参数  impurityParm= gini maxDepthParm= 3 maxBinsParm = 3. 所需时间=34 结果accuracy = 0.661330
训练评估：使用参数  impurityParm= gini maxDepthParm= 3 maxBinsParm = 5. 所需时间=7 结果accuracy = 0.677768
训练评估：使用参数  impurityParm= gini maxDepthParm= 3 maxBinsParm = 10. 所需时间=7 结果accuracy = 0.672066

训练评估：使用参数  impurityParm= entropy maxDepthParm= 15 maxBinsParm = 10. 所需时间=13 结果accuracy = 0.842933
训练评估：使用参数  impurityParm= entropy maxDepthParm= 15 maxBinsParm = 50. 所需时间=38 结果accuracy = 0.858378
调校后最佳参数  : impurity:entropy  ,maxDepth:15  ,maxBins:50  ,结果accuracy = 0.858378137735
==========测试阶段 ===============
使用test Data测试最佳模型，结果 accuracy:0.857969051217
========== 预测数据 ===============
共计：581012项
建立训练评估所需数据RDD...
土地条件 ：海拔:2596.0 方位:51.0 斜率:3.0  水源垂直距离:258.0  水源水平距离:0.0  9点时阴影:510.0....==>预测:4.0 实际:4.0结果:正确
土地条件 ：海拔:2590.0 方位:56.0 斜率:2.0  水源垂直距离:212.0  水源水平距离:-6.0  9点时阴影:390.0....==>预测:4.0 实际:4.0结果:正确
土地条件 ：海拔:2804.0 方位:139.0 斜率:9.0  水源垂直距离:268.0  水源水平距离:65.0  9点时阴影:3180.0....==>预测:1.0 实际:1.0结果:正确
```


图 17-15　找出最佳参数组合

17.9 运行 RunDecisionTreeMulti.py 不进行训练评估

因为训练评估比较费时，所以当我们已经找出最佳参数组合时就可以修改 trainEvaluateModel 程序为最佳参数组合。下次要进行预测时，可以不再运行训练评估，直接使用最佳参数进行预测即可。

步骤 01 修改 trainEvaluateModel 为最佳参数组合（见图 17-16）

```
*RunDecisionTreeMulti
if __name__ == "__main__":
    print("RunDecisionTreeMulti")
    sc=CreateSparkContext()
    print("========== 数据准备阶段 ===============")
    (trainData, validationData, testData) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("========== 训练评估阶段 ===============")
    (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \
        trainEvaluateModel(trainData, validationData, "entropy", 15,50)
    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
```


图 17-16　修改 trainEvaluateModel 为最佳参数组合

步骤 02 运行外部工具（见图 17-17）

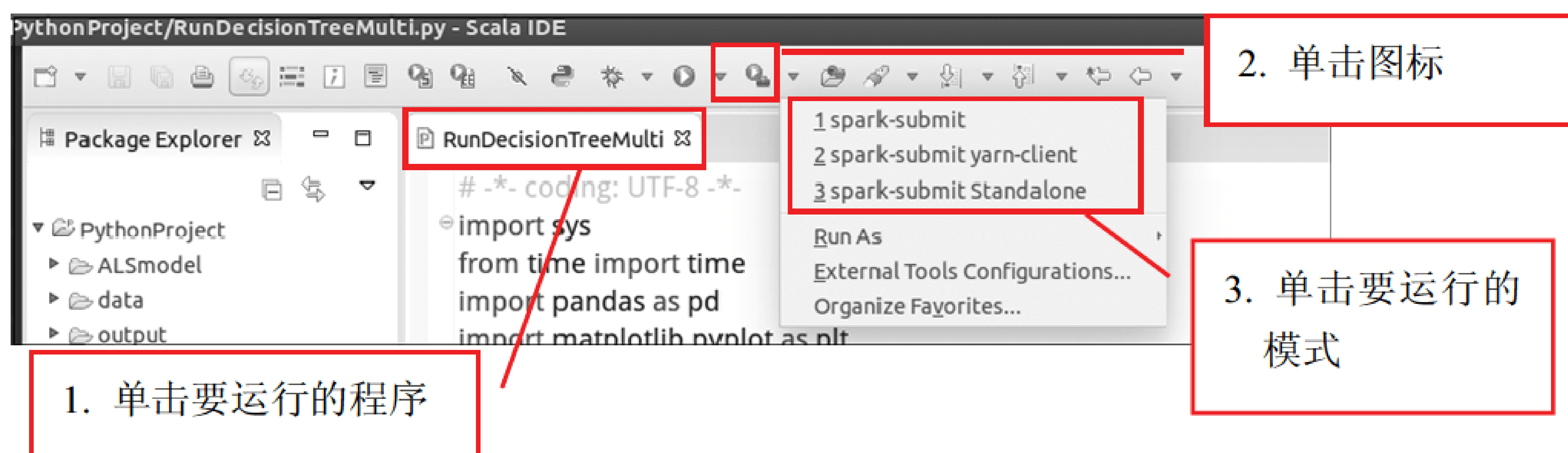


图 17-17　运行外部工具

步骤 03 不输入参数（见图 17-18）



图 17-18　不输入参数

步骤 04 运行结果

运行结果如图 17-19 所示。可以看到程序训练完成后就可以进行预测。因为不需要进行训练评估，所以运行所需的时间比较少。

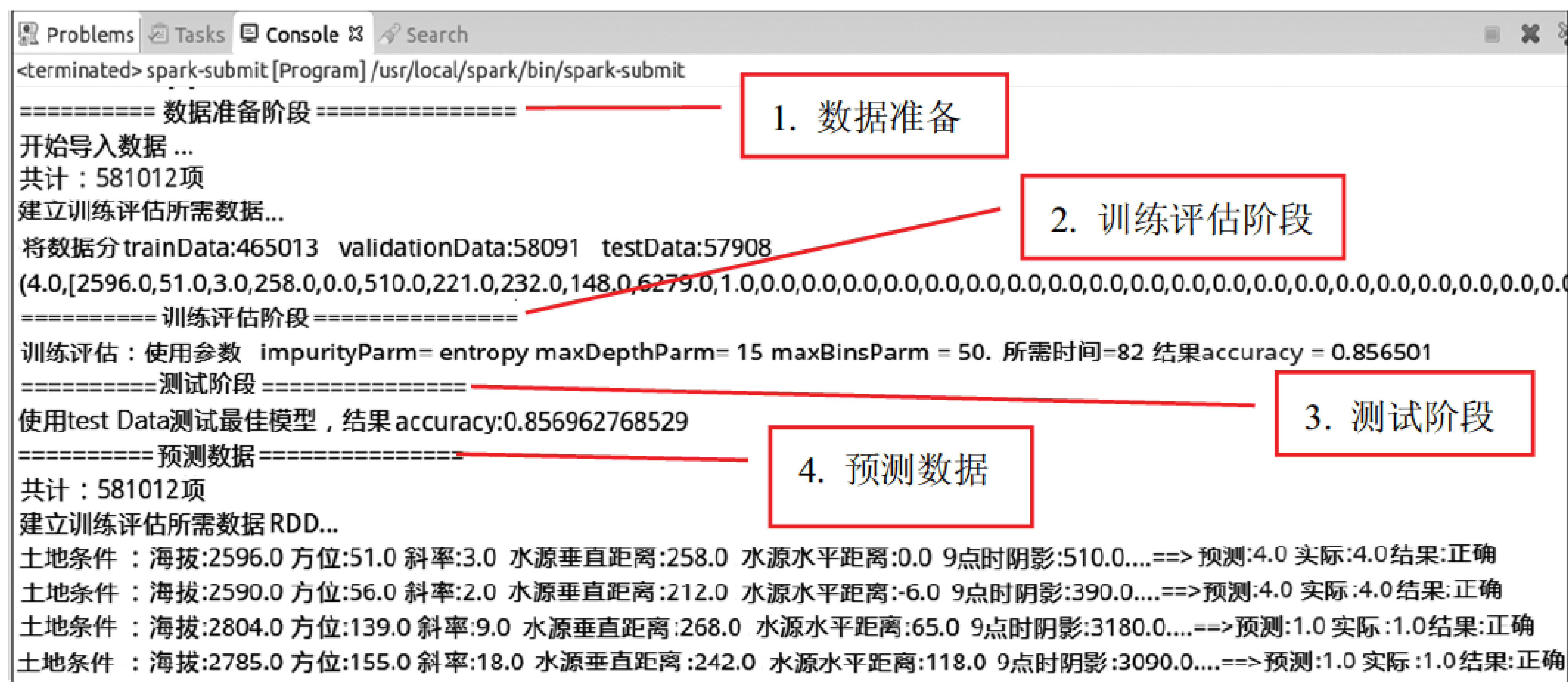


图 17-19　运行结果

（1）**数据准备阶段：**显示导入项数以及 trainData validationData testData 项数。

（2）**训练评估阶段：**进行训练评估，找出最佳参数组合，impurityParm = entropy，maxDepthParm= 15，maxBinsParm = 50，accuracy 大约为 0.85。

（3）**测试阶段：**使用测试数据再测试模型，accuracy 大约为 0.85，与训练阶段差异不大，确认没有 overfitting 的问题。

（4）**预测阶段：**使用 covtype.data 预测，显示在不同的土地条件下预测的值与实际的值，并显示预测结果是否准确。

17.10 结论

本章介绍了 Spark MLlib 决策树多元分类，完成了数据准备、训练模型，并通过训练评估找出了最佳参数组合，提高预测准确度。下一章我们将介绍决策树回归分析。

第 18 章 Python Spark MLlib 决策树回归分析

Bike Sharing（共享单车）系统就像城市的自行车出租系统，可以在某一地点租借，在另一个地点归还。本章将介绍决策树回归分析（DecisionTree Regression），预测在不同的情况（季节、月份、时间、假日、星期、工作日、天气、温度、体感温度、湿度、风速）下，每一小时的租用数量。

18.1 Bike Sharing 大数据问题分析

Bike Sharing（共享单车）系统的管理层希望能运用大数据分析提供更好的服务，因此找来了大数据分析师，与公司实际负责运营的工作人员组成一个团队，负责 Bike Sharing 大数据项目。

➢ 找出问题

“问对问题”是解决问题的第一步。大数据分析师与运营人员讨论后发现，运营人员面临一个问题，在不同的天气情况下，租用的数量差异很大。这就造成了困扰，运营人员常常无法提供足够数量的自行车，或是不知道何时进行维修工作。因此，他们希望能够预测在某种天气情况下的自行车租用数量。

预知自行车租用数量的好处如下：

- 自行车需要维修时，可以选择在自行车租用数量较少的时段。
- 在自行车租用数量大的时间，可以提供更多的自行车，增加营业额。

➢ 设计解决方案模型

大数据分析师与 Bike Sharing（共享单车）有实际工作经验的人员讨论哪些因素可能影响自行车租用的数量，经过讨论结果认为季节、月份、时间（0~23）、假日、星期、工作日、天气、温度、体感温度、湿度、风速都会影响自行车租借的数量。因此大数据分析师整理出表 18-1 所示的数据。

表 18-1　共享单车数据字段

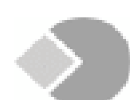
字段	说明
feature 特征	季节、月份、时间（0~23）、假日、星期、工作日、天气、温度、体感温度、湿度、风速
label 标签（预测目标）	每一小时的租用数量

➢ 搜集数据

- 每小时自行车租用数量可以在 Bike Sharing（共享单车）公司的计算机系统查询。
- 季节、月份、时间（0~23）、假日、星期、工作日，程序系统可以由每小时租用数量自动算出。
- 天气、温度、体感温度、湿度、风速这些天气的数据可以与提供天气历史数据的公司合作来获得。

➢ 分析数据

大数据分析师决定使用决策树回归分析来创建模型、训练、评估、预测数据。本章将详细

介绍如何进行决策树回归分析。

➢　**其他根据天气的预测应用**

事实上，这种根据天气预测的应用很广泛。例如，小连锁超市可以根据天气来预测关东煮、茶叶蛋、雨衣等的销售数量。如果可以预测销售数量，就可以事先准备足够的商品数量，从而增加销售额，也可以减少因为准备过多的商品而导致食材的浪费或者商品的积压。除了小连锁超市，餐厅也很适合使用天气来预测销售数量。

18.2　Bike Sharing 数据集

在本章中不需要自行搜集这些数据，有研究单位已经创建了这些数据。可以使用浏览器来下载这些数据，网址为 https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset。

图 18-1 即为 Bike Sharing 数据集的下载网站。

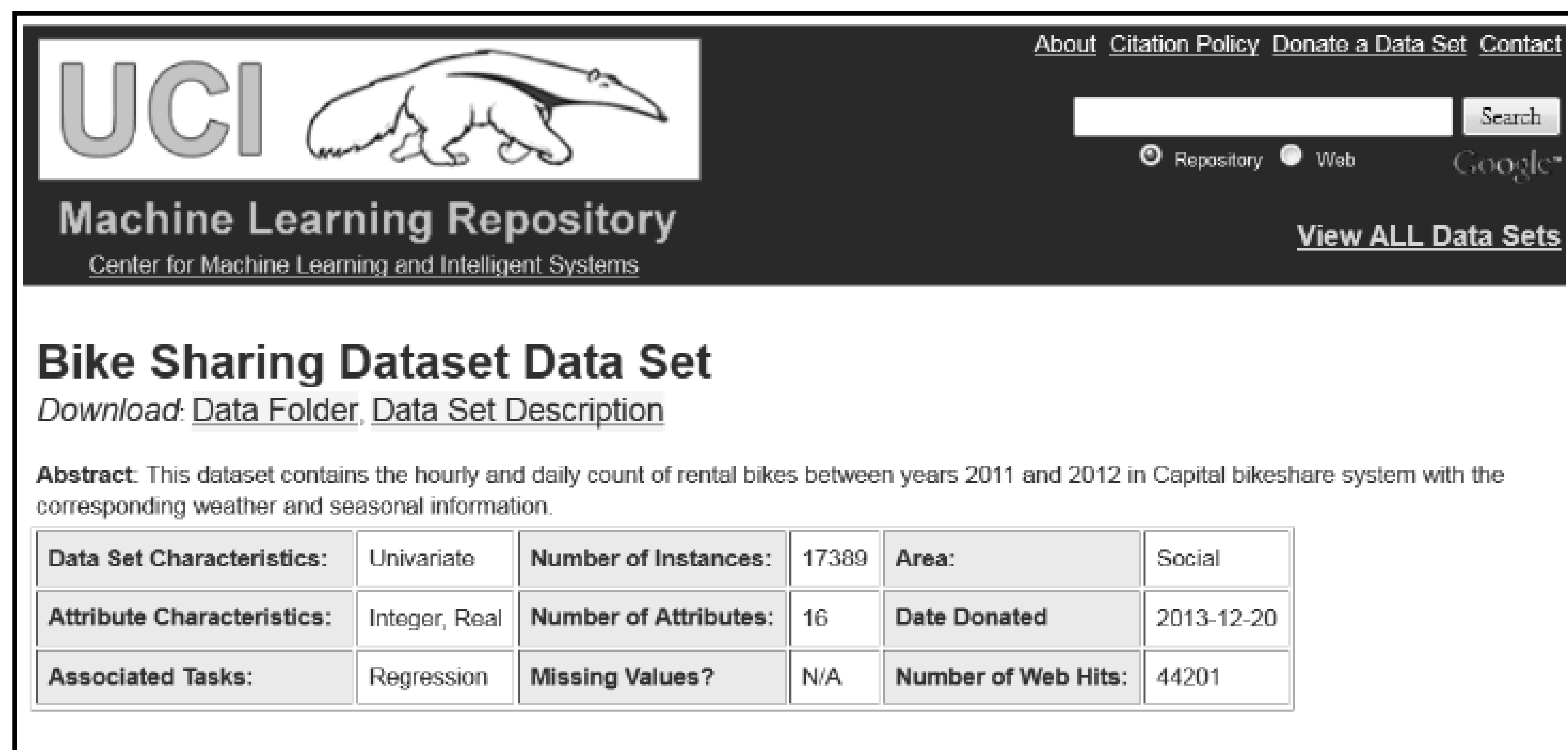


图 18-1　Bike Sharing 数据集的下载网站

18.3　下载与查看数据

步骤 01　下载/解压缩文件

在“终端”程序中输入下述命令：

➢　**切换到项目数据目录**

```
cd ~/workspace/Classification/data
```

➢　**下载** Bike-Sharing-Dataset.zip

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-
Sharing-Dataset.zip
```

- **解压缩 Bike-Sharing-Dataset.zip**

```
unzip -j Bike-Sharing-Dataset.zip
```

运行后屏幕显示界面如图 18-2 所示。

```
hduser@master: ~/workspace/Classification/data
hduser@master:~$ cd ~/workspace/Classification/data
hduser@master:~/workspace/Classification/data$ wget https://archive.ics.uci.edu/
ml/machine-learning-databases/00275/Bike-Sharing-Dataset.zip
--2016-05-18 22:14:17--  https://archive.ics.uci.edu/ml/machine-learning-databas
es/00275/Bike-Sharing-Dataset.zip
正在解析主机 archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.249
正在连接 archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.249|:443... 已连接
。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 279992 (273K) [application/zip]
正在保存至: “Bike-Sharing-Dataset.zip”

100%[===================================>] 279,992     44.0KB/s   用时 6.2s

2016-05-18 22:14:25 (44.0 KB/s) - 已保存 “Bike-Sharing-Dataset.zip” [279992/2799
92])

hduser@master:~/workspace/Classification/data$ unzip -j Bike-Sharing-Dataset.zip
Archive:  Bike-Sharing-Dataset.zip
  inflating: Readme.txt
  inflating: day.csv
  inflating: hour.csv
hduser@master:~/workspace/Classification/data$
```

图 18-2　下载/解压缩文件

步骤 02 查看 Readme.txt

在“终端”程序中输入下述命令：

- **查看说明文件（见图 18-3）**

```
cat Readme.txt|more
```

在目录下主要有两个数据文件：

- hour.csv——以小时为单位求和（租借数量）。
- day.csv——以天为单位求和（租借数量）。

```
hduser@master: ~/workspace/DecisionTree/data
Files
=========================================

	- Readme.txt
	- hour.csv : bike sharing counts aggregated on hourly basis. Records: 17379 hours
	- day.csv - bike sharing counts aggregated on daily basis. Records: 731 days


=========================================
Dataset characteristics
=========================================
Both hour.csv and day.csv have the following fields, except hr which is not available in day.csv

	- instant: record index
	- dteday : date
	- season : season (1:springer, 2:summer, 3:fall, 4:winter)
	- yr : year (0: 2011, 1:2012)
	- mnth : month ( 1 to 12)
	- hr : hour (0 to 23)
	- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
	- weekday : day of the week
	- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
	+ weathersit :
		- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
		- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
		- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
		- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
	- temp : Normalized temperature in Celsius. The values are divided to 41 (max)
	- atemp: Normalized feeling temperature in Celsius. The values are divided to 50 (max)
	- hum: Normalized humidity. The values are divided to 100 (max)
	- windspeed: Normalized wind speed. The values are divided to 67 (max)
	- casual: count of casual users
	- registered: count of registered users
	- cnt: count of total rental bikes including both casual and registered
```

图 18-3　查看说明文件

步骤 03　双击 hour.csv 打开该文件（见图 18-4）

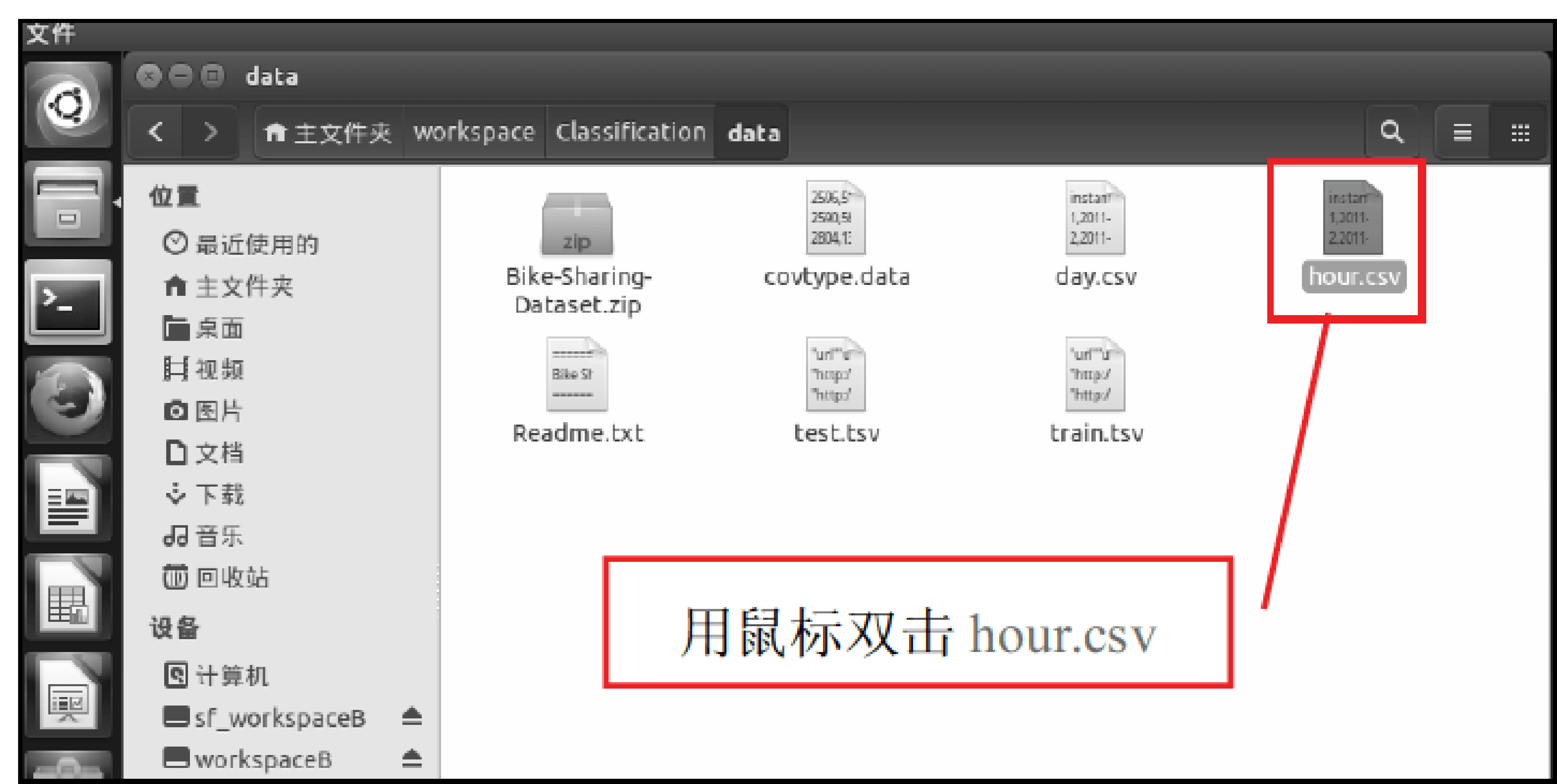


图 18-4　打开 hour.csv 文件

步骤 04　查看 hour.csv（见图 18-5）

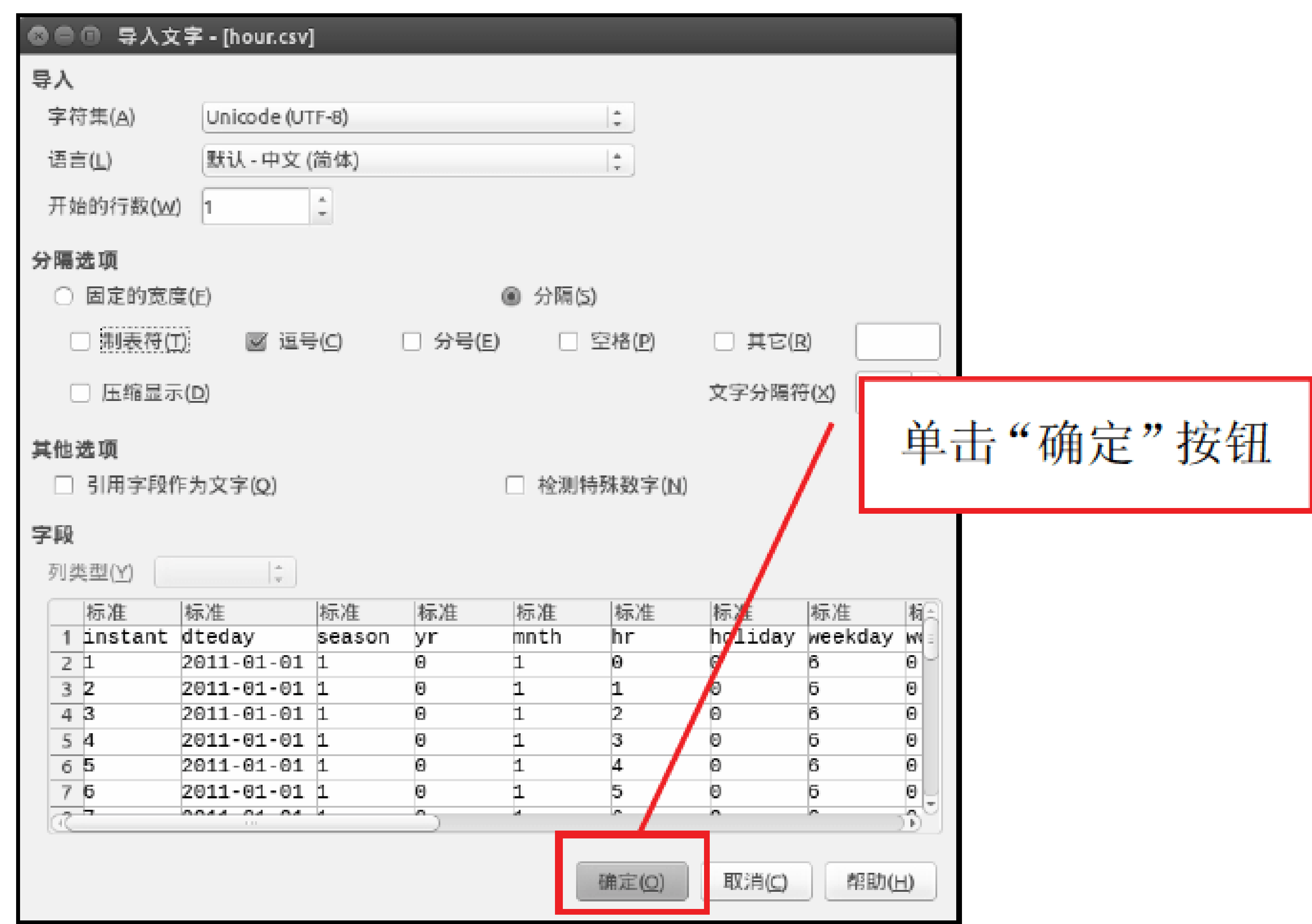


图 18-5　查看 hour.csv

步骤 05　hour.csv 字段介绍（见图 18-6）

features 字段

label 字段

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	asual	registered	cnt
2	1	2011-01-0	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
3	2	2011-01-0	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
4	3	2011-01-0	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
5	4	2011-01-0	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
6	5	2011-01-0	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	1	1
7	6	2011-01-0	1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	0	1	1
8	7	2011-01-0	1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2	0	2
9	8	2011-01-0	1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	1	2	3
10	9	2011-01-0	1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	1	7	8
11	10	2011-01-0	1	0	1	9	0	6	0	1	0.32	0.3485	0.76	0	8	6	14
12	11	2011-01-0	1	0	1	10	0	6	0	1	0.38	0.3939	0.76	0.2537	12	24	36
13	12	2011-01-0	1	0	1	11	0	6	0	1	0.36	0.3333	0.81	0.2836	26	30	56
14	13	2011-01-0	1	0	1	12	0	6	0	1	0.42	0.4242	0.77	0.2836	29	55	84
15	14	2011-01-0	1	0	1	13	0	6	0	2	0.46	0.4545	0.72	0.2985	47	47	94

图 18-6　hour.csv 字段

hour.csv 字段说明如表 18-2 所示。我们可以先思考一下如何处理。

表 18-2　hour.csv 字段说明

字段	说明	处理方式
instant	序号 ID	忽略
dteday	日期	忽略
season	季节（1:springer, 2:summer, 3:fall, 4:winter）	特征字段
yr	年份（0:2011,1:2012）	忽略
mnth	月份（1~12）	特征字段
hr	时间（0~23）	特征字段
holiday	假日：0 为非假日、1 为假日	特征字段
weekday	星期	特征字段
workingday	工作日，非周末而且非假日	特征字段
weathersit	天气： 1：Clear, Few clouds, Partly cloudy, Partly cloudy 2：Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3：Light Snow, Light Rain+Thunderstorm+Scattered clouds,Light Rain+Scattered clouds 4：Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	特征字段
temp	摄氏温度，将温度除以 41 进行标准化	特征字段
atemp	实际感觉温度，将温度除以 50 进行标准化	特征字段
hum	湿度，将湿度除以 100 进行标准化	特征字段
windspeed	风速，将风速除以 67 进行标准化	特征字段
casual	临时用户，此时段租借自行车的数量	忽略
registered	登录会员，此时段租借的数目	忽略
cnt	此时段租借总数量 cnt= casual+ registered	标签字段（预测目标）

以上部分字段会忽略，不列入特征字段：

- instant（序号 ID）、dteday（日期）与预测的结果无关，所以忽略这两个字段。
- yr 年份（0:2011,1:2012），因为我们希望预测未来的年份，所以忽略此字段。
- casual 与 registered，因为 casual 加 registered 等于 cnt（标签字段）。这两个字段已经隐含在标签字段信息里，所以忽略这两个字段。

18.4 修改 PrepareData() 数据准备

决策树回归分析 RunDecisionTreeRegression.py 完整的程序代码请参考本书范例程序。决策树回归分析程序的基本架构与第 13 章介绍的决策树二元分析 RunDecisionTreeBinary.py 类似，因而我们可以参考第 13 章的详细说明。本章仅说明 RunDecisionTreeRegression.py 重要的修改

部分。

步骤01 修改 PrepareData() 数据准备

PrepareData 函数传入的参数 sc 是 SparkContext，返回 trainData、validationData、testData，主要分为下列 3 个步骤：

```
def PrepareData(sc):
    #---------1. 导入并转换数据 -------------
    #---------2. 建立训练评估所需数据 RDD[LabeledPoint]---------
    #---------3. 以随机方式将数据分为 3 个部分并且返回 -----------
```

步骤02 导入并转换数据

输入下列程序代码：

```
#---------------------1. 导入并转换数据 -------------
print(" 开始导入数据 ...")
rawDataWithHeader = sc.textFile(Path+"data/hour.csv")
header = rawDataWithHeader.first()
rawData = rawDataWithHeader.filter(lambda x:x !=header)
lines = rawData.map(lambda x: x.split(","))
print (lines.first())
print(" 共计：" + str(lines.count()) + " 项 ")
```

以上程序代码说明如下：

（1）使用 sc.textFile 读取 hour.csv。

（2）header = rawDataWithHeader.first() 取得第 1 行。

（3）第 1 行表头是字段名，所以使用下列程序代码删除 rawDataWithHeader.filter(lambda x:x!=header)。

（4）lines =rawData.map(lambda x: x.split(","))以 "," 间隔取出每一个字段。

（5）打印第一项数据与数据项数。

步骤03 建立训练评估所需数据 RDD[LabeledPoint]

以下程序代码使用 map 对每一项数据进行提取标签字段与特征字段。建立 LabelPoint，作为后续训练数据。

- 提取标签字段：extract_label(r)，步骤 4 会介绍。
- 提取特征字段：extract_features(r,len(r) - 1))，步骤 5 会介绍。

```
#---------2. 建立训练评估所需数据 RDD[LabeledPoint]---------
labelpointRDD = lines.map(lambda r:LabeledPoint(
                         extract_label(r),
                         extract_features(r,len(r) - 1)))
```

```
print labelpointRDD.first()
```

步骤04 extract_label () 提取 label 字段

以下 extract_label () 程序代码用于提取 label 字段：

```
def extract_label(record):
    label=(record[-1])
    return float(label)
```

以上程序代码说明如下：

（1）先使用 label=(record[-1])取出最后一个字段，就是 label 标签字段。

（2）再使用 return float(label)返回 label 转换为 float。

步骤05 extract_feature() 提取特征字段

以下 extract_feature() 程序代码用于提取特征字段：

```
def extract_features(record,featureEnd):
featureSeason=[convert_float(field) for field in record[2]]
features=[convert_float(field) for field in record[4: featureEnd-2]]
return np.concatenate( (featureSeason, features))
```

以上程序代码说明如下：

（1）featureSeason=[convert_float(field) for field in record[2]] 提取第 2 个字段（从 0 算起）season，执行 convert_float 函数，转换为 float。

（2）features= [convert_float(field) for field in record[4: featureEnd-2]]，因为我们会忽略 yr、casual、registered 字段，所以从第 4 个字段（从 0 算起）读取到 featureEnd-2 字段，执行 convert_float 函数，转换为 float。

（3）最后返回 np.concatenate((featureSeason, features))。

步骤06 以随机方式将数据分为 3 部分并返回

以下程序代码 .randomSplit([8, 1, 1]) 按照 8:1:1 的比例以随机方式分为 3 个部分（trainData, validationData, testData）并返回数据。

```
#-----------3. 以随机方式将数据分为 3 个部分并且返回 -----------
(trainData, validationData, testData) = labelpointRDD.randomSplit([8, 1, 1])
print(" 将数据分 trainData:" + str(trainData.count()) +
"validationData:" + str(validationData.count()) +
"testData:" + str(testData.count()))
return (trainData, validationData, testData) # 返回数据
```

18.5 修改 DecisionTree.trainRegressor 训练模型

在决策树回归分析时，我们必须使用 DecisionTree.trainRegressor。决策树回归分析的参数设置：无 numClasses 参数，而且 impurity 固定是 "variance"。

步骤 01 修改 trainEvaluateModel 参数评估（见图 18-7）

```
def trainEvaluateModel(trainData,validationData,
                       impurityParm, maxDepthParm, maxBinsParm):
    startTime = time()
    model = DecisionTree.trainRegressor(trainData,
                   categoricalFeaturesInfo={}, \
                   impurity=impurityParm,
                   maxDepth=maxDepthParm,
                   maxBins=maxBinsParm)
    RMSE = evaluateModel(model, validationData)
    duration = time() - startTime
    print    " 训练评估：使用参数 " + \
        " impurityParm= %s"%impurityParm+ \
        " maxDepthParm= %s"%maxDepthParm+ \
        " maxBinsParm = %d."%maxBinsParm + \
        " 所需时间=%d"%duration + \
        " 结果 RMSE = %f " % RMSE
    return (RMSE,duration, impurityParm, maxDepthParm, maxBinsParm,model)
```

修改为 DecisionTree.trainRegressor

图 18-7　修改 trainEvaluateModel 参数评估

步骤 02 修改 parametersEval() 参数评估

在决策树回归分析中，必须使用 DecisionTree.trainRegressor 训练数据，其中 impurity 固定是 "variance"，因此将 impurityList 参数改为 ["variance"]，如图 18-8 所示。

```
RunDecisionTreeRegression
def parametersEval(training_RDD, validation_RDD):

    print("----- 评估 maxDepth 参数使用 ---------")
    evalParameter(training_RDD, validation_RDD,"maxDepth",
                  impurityList=["variance"],
                  maxDepthList =[3, 5, 10, 15, 20, 25] ,
                  maxBinsList=[10])
    print("----- 评估 maxBins 参数使用 ---------")
    evalParameter(training_RDD, validation_RDD,"maxBins",
                  impurityList=["variance"],
                  maxDepthList=[10],
                  maxBinsList=[3, 5, 10, 50, 100, 200 ])
```

固定为 variance

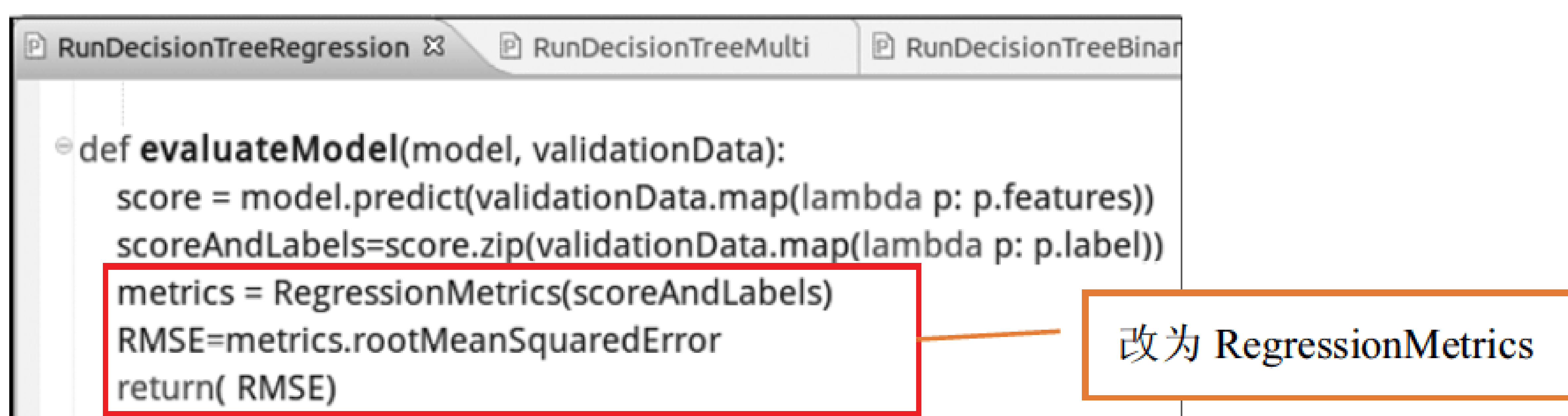
图 18-8　修改 parametersEval() 参数评估

18.6 以 RMSE 评估模型准确率

在回归分析（Regression ）中，我们使用 RMSE（Root Mean Square Error）“计算预测的结果”与“标签字段”的误差平均值。通常 RMSE 越小，代表误差越小，即代表预测值与真实的值越接近，表示准确度越高。

步骤 01 修改 evaluateModel 评估模型计算 RMSE

我们可以使用 RegressionMetrics 先建立 Metrics，然后使用 .rootMeanSquaredError 方法计 算均方根差（RMSE）。修改 evaluateModel 程序代码，计算 RMSE，如图 18-9 所示。



```
def evaluateModel(model, validationData):
    score = model.predict(validationData.map(lambda p: p.features))
    scoreAndLabels=score.zip(validationData.map(lambda p: p.label))
    metrics = RegressionMetrics(scoreAndLabels)
    RMSE=metrics.rootMeanSquaredError
    return( RMSE)
```


图 18-9 修改 evaluateModel 评估模型计算 RMSE

步骤 02 修改 evaluateParameter 评估参数

将程序代码改为 RMSE 评估参数，如图 18-10 所示。



```
def evalParameter(trainData, validationData, evaparm,impurityList, maxDepthList, maxBinsList):
    metrics = [trainEvaluateModel(trainData, validationData,  impurity,maxdepth,  maxBins  )
                    for impurity in impurityList
                    for maxdepth in maxDepthList
                    for maxBins in maxBinsList ]
    if evaparm=="impurity":
        IndexList=impurityList[:]
    elif evaparm=="maxDepth":
        IndexList=maxDepthList[:]
    elif evaparm=="maxBins":
        IndexList=maxBinsList[:]

    df = pd.DataFrame(metrics,index=IndexList,
            columns=['RMSE', 'duration','impurityParm', 'maxDepthParm', 'maxBinsParm','model'])
    ax = df['RMSE'].plot(kind='bar', title =evaparm,figsize=(10,6),legend=True, fontsize=12)
    ax.set_xlabel(evaparm,fontsize=12)
    ax.set_xlabel
    ax.set_ylim([0,200])
    ax.set_ylabel('RMSE',fontsize=12)
    ax2 = ax.twinx()
    ax2.plot(df[['duration']].values, linestyle='-', marker='o', linewidth=2.0,color='r')
    plt.show()
```


图 18-10 修改 evaluateParameter 评估参数

18.7 训练评估找出最好的参数组合

evaluateAllParameter 程序将 3 个参数训练评估，找出最好的参数组合。

在回归分析中，我们以 RMSE 作为评估模型的准确率。RMSE 越小，代表误差越小，准确率越高，所以使用从小到大排序，并取出第一项数据，找出最佳的模型（可参考图 18-11 中的程序代码）。

```
def evalAllParameter(training_RDD, validation_RDD, impurityList, maxDepthList, maxBinsList):
    metrics = [trainEvaluateModel(trainData, validationData,  impurity,maxdepth,  maxBins  )
                    for impurity in impurityList
                    for maxdepth in maxDepthList
                    for maxBins in maxBinsList ]
    Smetrics = sorted(metrics, key=lambda k: k[0])
    bestParameter=Smetrics[0]

    print(" 调校后最佳参数：impurity:" + str(bestParameter[2]) +
          " ,maxDepth:" + str(bestParameter[3]) +
          " ,maxBins:" + str(bestParameter[4])  +
          " ,结果RMSE = " + str(bestParameter[0]))

    return bestParameter[5]
```

排序改为从小到大

图 18-11　找出最好的参数组合

18.8 使用训练完成的模型预测数据

步骤 01 修改 PredictData()预测数据

PredictData() 函数包括下列 4 个步骤：

```
def PredictData(sc,model):
    #----------------------1. 导入并转换数据 -------------
    #----------------------2. 建立预测所需数据 LabeledPoint RDD-------------
    #----------------------3. 定义字典 ----------------
    #----------------------4. 进行预测并显示结果 --------------
```

第 1、2 步与 PrepareData() 相同，可自行参考第 18.4 节的内容，第 3 步定义字典与第 4 步进行预测并显示结果将在后文介绍。

步骤 02 定义字典

后续我们希望能将数字显示成有意义的文字，所以定义下列字典：

```
#----------------------3. 定义字典 ----------------
SeasonDict = { 1 : " 春 ", 2 : " 夏 ", 3 :" 秋 ", 4 : " 冬 " }
HoildayDict={ 0 : " 非假日 ", 1 : " 假日 " }
WeekDict = {0:" 一 ",1:" 二 ",2:" 三 ",3:" 四 ",4 :" 五 ",5:" 六 ",6:" 日 "}
WorkDayDict={ 1 : " 工作日 ", 0 : " 非工作日 " }
WeatherDict={ 1 : " 晴 ", 2 : " 阴 ", 3 : " 小雨 ", 4 : " 大雨 " }
```

步骤03 进行预测并显示结果

输入下列程序代码：

```
#----------------------4. 进行预测并显示结果 -------------
for lp in labelpointRDD.take(20):
predict = int(model.predict(lp.features))
label=lp.label
features=lp.features
result = (" 正确 " if (label == predict) else " 错误 ")
error = math.fabs(label - predict)
dataDesc=" 特征 : "+SeasonDict[features[0]] +" 季 ,"+\
str(features[1]) + " 月 ," +str(features[2]) + " 时 ,"+ \
HoildayDict[features[3]] +","+" 星期 "+WeekDict[features[4]]+","+ \
WorkDayDict[features[5]]+","+ WeatherDict[features[6]]+","+\
str(features[7] 41)+ " 度 ,"+" 体感 " + str(features[8] 50) + " 度 ," +\
" 湿度 " + str(features[9] 100) + ","+ " 风速 " + str(features[10] 67) +\
" ==> 预测结果 :" + str(predict )+\
" , 实际 :" + str(label) + result + ", 误差 :" + str(error)
print dataDesc
```

以上程序代码说明如表 18-3 所示。

表 18-3　程序代码说明

程序代码	说明
for lp in labelpointRDD.take(20):	取出 20 项进行预测
predict = int(model.predict(lp. features))	使用训练完成的模型传入特征 lp.features 进行预测，预测结果存放在 predict 变量中
label=lp.label	取得标签字段
features=lp.features	取得特征字段
result = (" 正 确 " if (label == predict) else " 错误 ")	判断预测是否正确
error = math.fabs(label - predict)	将 label（标签字段） 与 predict（预测结果）相减，得出误差
dataDesc=" 特征 :"+	以下特征的说明都存放在 dataDesc 变量中
SeasonDict[features[0]] +" 季 ,"+\	显示季节
str(features[1]) + " 月 ," +\	显示月份

（续表）

程序代码	说明
str(features[2]) + " 时 ,"+ \	显示时间
HoildayDict[features[3]] +","+\	使用字典 HoildayDict，显示假日
" 星期 "+WeekDict[features[4]]+","+ \	使用字典 WeekDict，显示星期
WorkDayDict[features[5]]+","+\	使用字典 WorkDayDict，显示工作日 / 非工作日
WeatherDict[features[6]]+","+\	使用字典 WeatherDict，显示天气
str(features[7] 41)+ " 度 ,"+\	显示的气温，因为原始数据已经将温度除以 41 进行了标准化，所以再乘以 41 还原真实的温度
"体感" + str(features[8] 50) + "度," +\	实际感觉温度，因为原始数据已经将感觉温度除以 50 进行了标准化，所以再乘以 50 还原真实的温度
" 风速 " + str(features[10] 67) +\	因为原始数据已经将风速除以 67 进行了标准化，所以再乘以 67 还原真实的风速
" ==> 预测结果 :" + str(predict)+\	显示预测结果
" ，实际 :" + str(label) + result +	显示实际数据与结果
"，误差 :" + str(error)	显示误差
print dataDesc	显示 dataDesc 变量全部结果

18.9 运行RunDecisionTreeMulti.py进行参数评估

步骤 01 运行外部工具（见图 18-12）

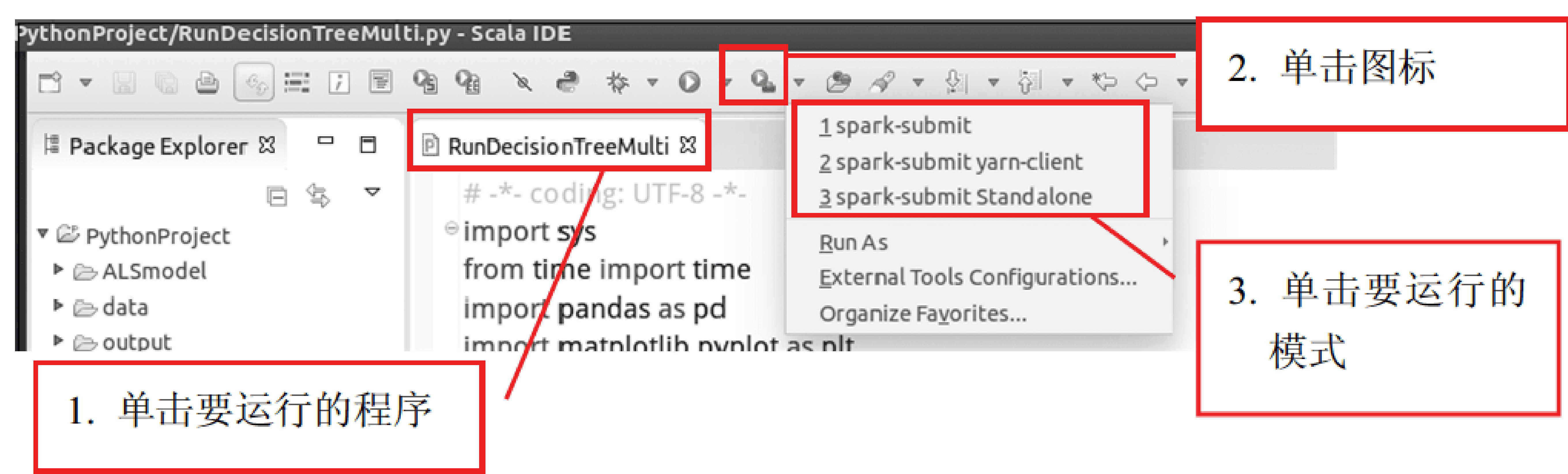


图 18-12　运行外部工具

步骤02 输入参数“-e”（见图 18-13）



图 18-13　输入参数“-e”

步骤03 maxDepth 参数评估（见图 18-14）

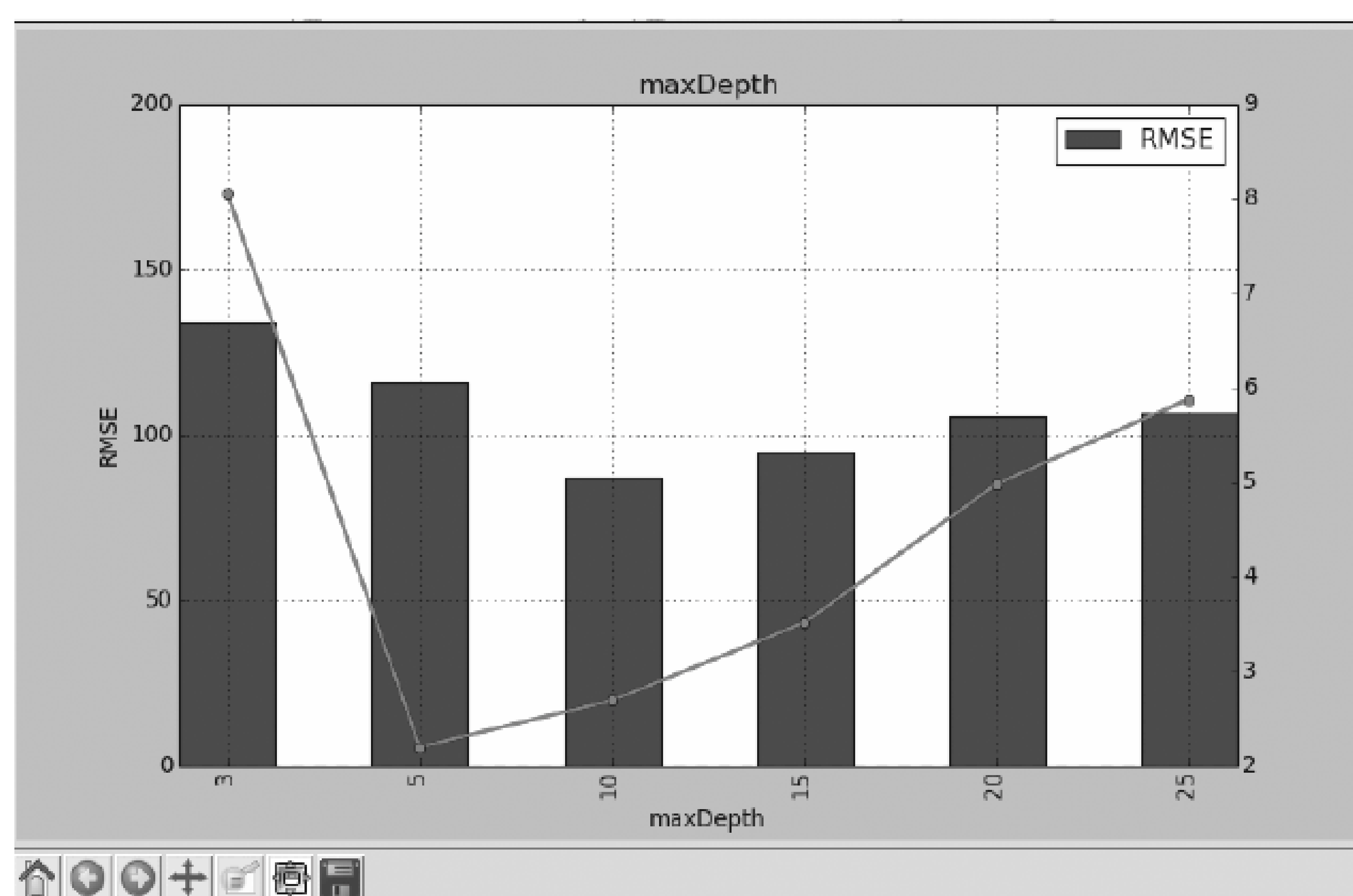


图 18-14　maxDepth 参数评估

从图 18-14 可以看到 maxDepth=10 时 RMSE 最低，maxDepth 越大，所需时间越多，所以 maxDepth=10 是不错的选择。

步骤04 maxBins 参数评估（见图 18-15）

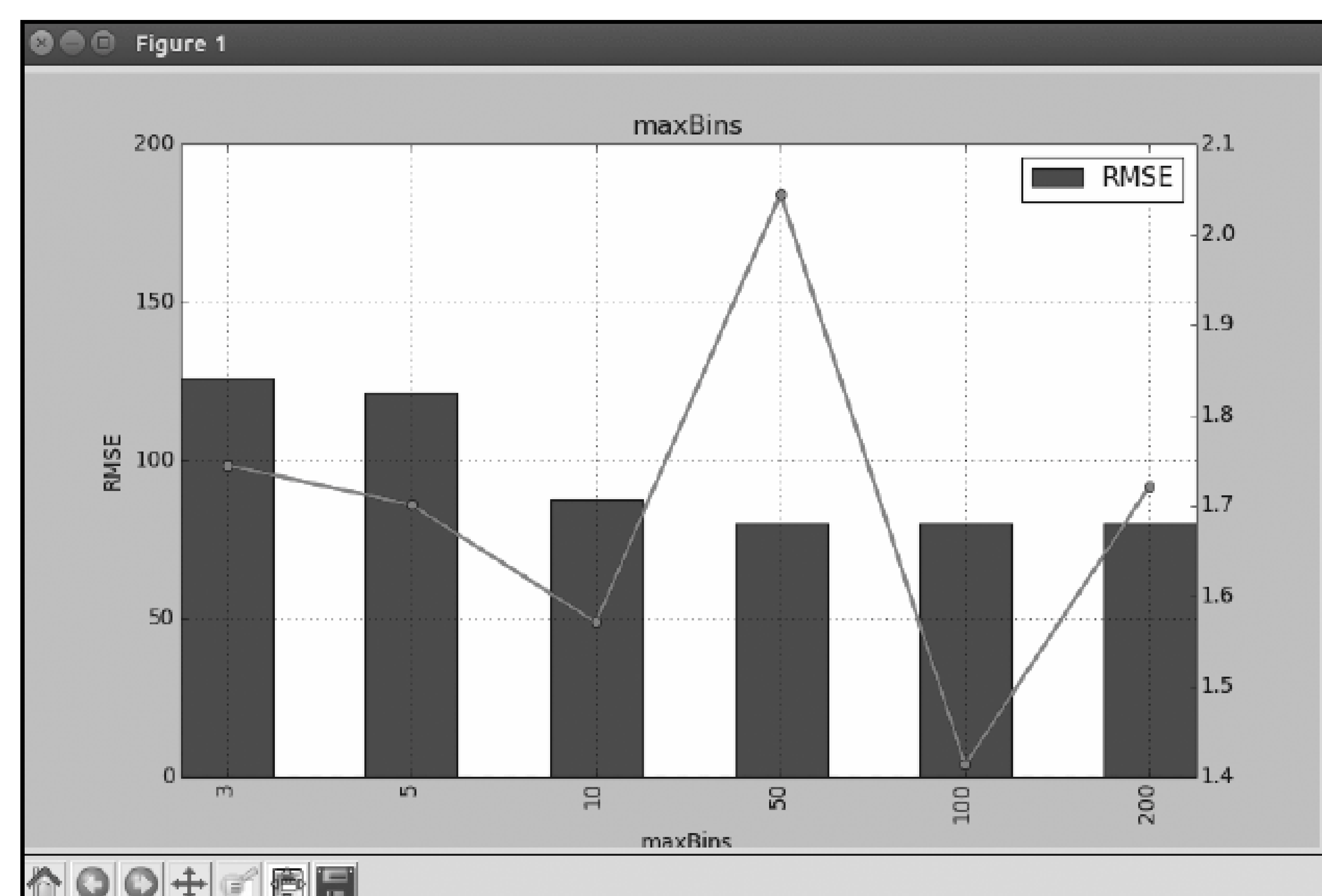


图 18-15　maxBins 参数评估



从图 18-15 可以看到 maxBins=200 时 RMSE 最低，maxBins 越大，所需的时间越多，所以 maxBins=200 是不错的选择。

18.10 运行 RunDecisionTreeMulti.py 训练评估参数并找出最好的参数组合

接下来，我们用 evaluateAllParameter 函数训练评估 3 种参数，希望找出最好的参数组合。

步骤 01 运行外部工具（见图 18-16）

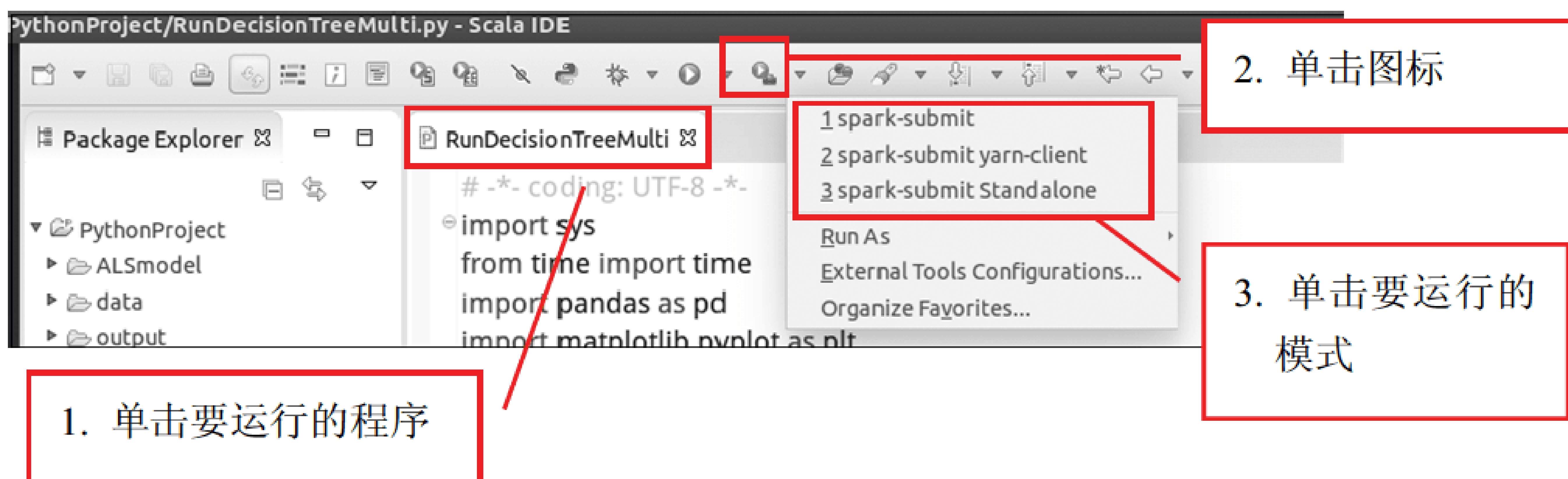


图 18-16　运行外部工具

步骤 02 输入参数“-a”（见图 18-17）



图 18-17　输入参数“-a”

步骤 03 进行训练评估找出最佳参数组合（见图 18-18）

```
Problems  Tasks  Console  Search
<terminated> spark-submit [Program] /usr/local/spark/bin/spark-submit
16/08/31 09:55:17 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classe
master=local[*]
========== 数据准备阶段 ===============
开始导入数据 ...
[u'1', u'2011-01-01', u'1', u'0', u'1', u'0', u'0', u'6', u'0', u'1', u'0.24', u'0.2879', u'0.81', u'0', u'3', u'13', u'16']
共计：17379项
(16.0,[1.0,1.0,0.0,0.0,6.0,0.0,1.0,0.24,0.2879,0.81,0.0])
将数据分trainData:13918  validationData:1712  testData:1749
========== 训练评估阶段 ===============
训练评估：使用参数  impurityParm= variance  maxDepthParm= 10  maxBinsParm = 100.  所需时间=11  结果RMSE = 75.874122
----- 所有参数训练评估找出最好的参数组合 ---------
训练评估：使用参数  impurityParm= variance  maxDepthParm= 3  maxBinsParm = 3.  所需时间=1  结果RMSE = 138.628679
训练评估：使用参数  impurityParm= variance  maxDepthParm= 3  maxBinsParm = 5.  所需时间=1  结果RMSE = 143.229976
训练评估：使用参数  impurityParm= variance  maxDepthParm= 3  maxBinsParm = 10.  所需时间=0  结果RMSE = 132.609611

训练评估：使用参数  impurityParm= variance  maxDepthParm= 25  maxBinsParm = 100.  所需时间=5  结果 RMSE = 92.814461
训练评估：使用参数  impurityParm= variance  maxDepthParm= 25  maxBinsParm = 200.  所需时间=5  结果 RMSE = 92.814461
调校后最佳参数  : impurity:variance  ,maxDepth:10  ,maxBins:100  ,结果RMSE = 75.8741217226
==========测试阶段 ===============
使用test Data测试最佳模型，结果RMSE:80.0462945993
========== 预测数据 ===============
开始导入数据 ...
共计：17379项
 特征：春季,1.0月,0.0时,非假日,星期日,非工作日,晴,9.84度,体感14.395度,湿度81.0,风速0.0 ==> 预测结果:20 ，实际:16.0错误,  误差:4.0
 特征：春季,1.0月,1.0时,非假日,星期日,非工作日,晴,9.02度,体感13.635度,湿度80.0,风速0.0 ==> 预测结果:51 ，实际:40.0错误,  误差:11.0
```

进行训练评估找出最佳参数组合

最佳参数

图 18-18　找出最佳参数组合

18.11 运行 RunDecisionTreeMulti.py 不进行训练评估

因为训练评估比较费时，所以当我们已经找出最佳参数组合时就可以修改 trainEvaluateModel 程序为最佳参数组合。下次要进行预测时，可以不再进行训练评估，直接使用最佳参数进行预测。

步骤 01 修改 trainEvaluate 为最佳参数组合（见图 18-19）

```
if __name__ == "__main__":
    print("RunDecisionTreeRegression")
    sc=CreateSparkContext()
    print("==========数据准备阶段 ===============")
    (trainData, validationData, testData) =PrepareData(sc)
    trainData.persist(); validationData.persist(); testData.persist()
    print("========== 训练评估阶段 ===============")
    (AUC,duration, impurityParm, maxDepthParm, maxBinsParm,model)= \
        trainEvaluateModel(trainData, validationData, "variance", 10, 100)
    if (len(sys.argv) == 2) and (sys.argv[1]=="-e"):
        parametersEval(trainData, validationData)
    elif  (len(sys.argv) == 2) and (sys.argv[1]=="-a"):
```

修改为最佳参数组合

图 18-19　修改 trainEvaluate 为最佳参数组合

步骤 02　执行外部工具（见图 18-20）

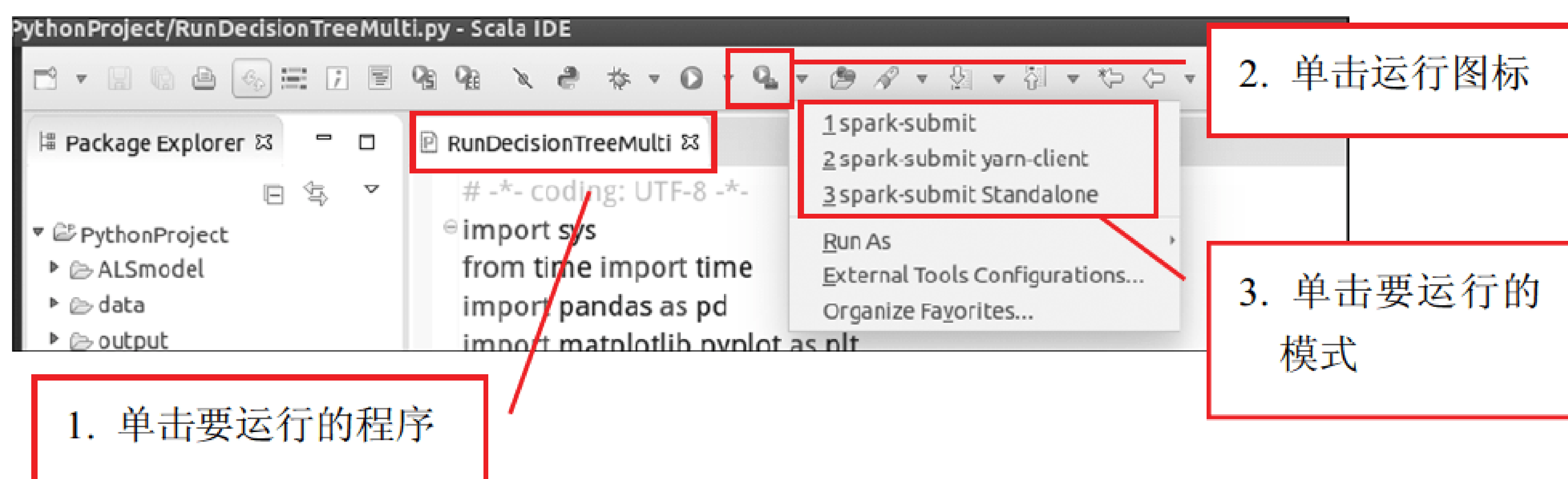


图 18-20　执行外部工具

步骤 03　不输入参数（见图 18-21）



图 18-21　不输入参数

步骤 04　运行结果

运行结果如图 18-22 所示，从中可以看到程序训练完成后就能够进行预测。因为不需要进行训练评估，所以运行所需的时间比较少。

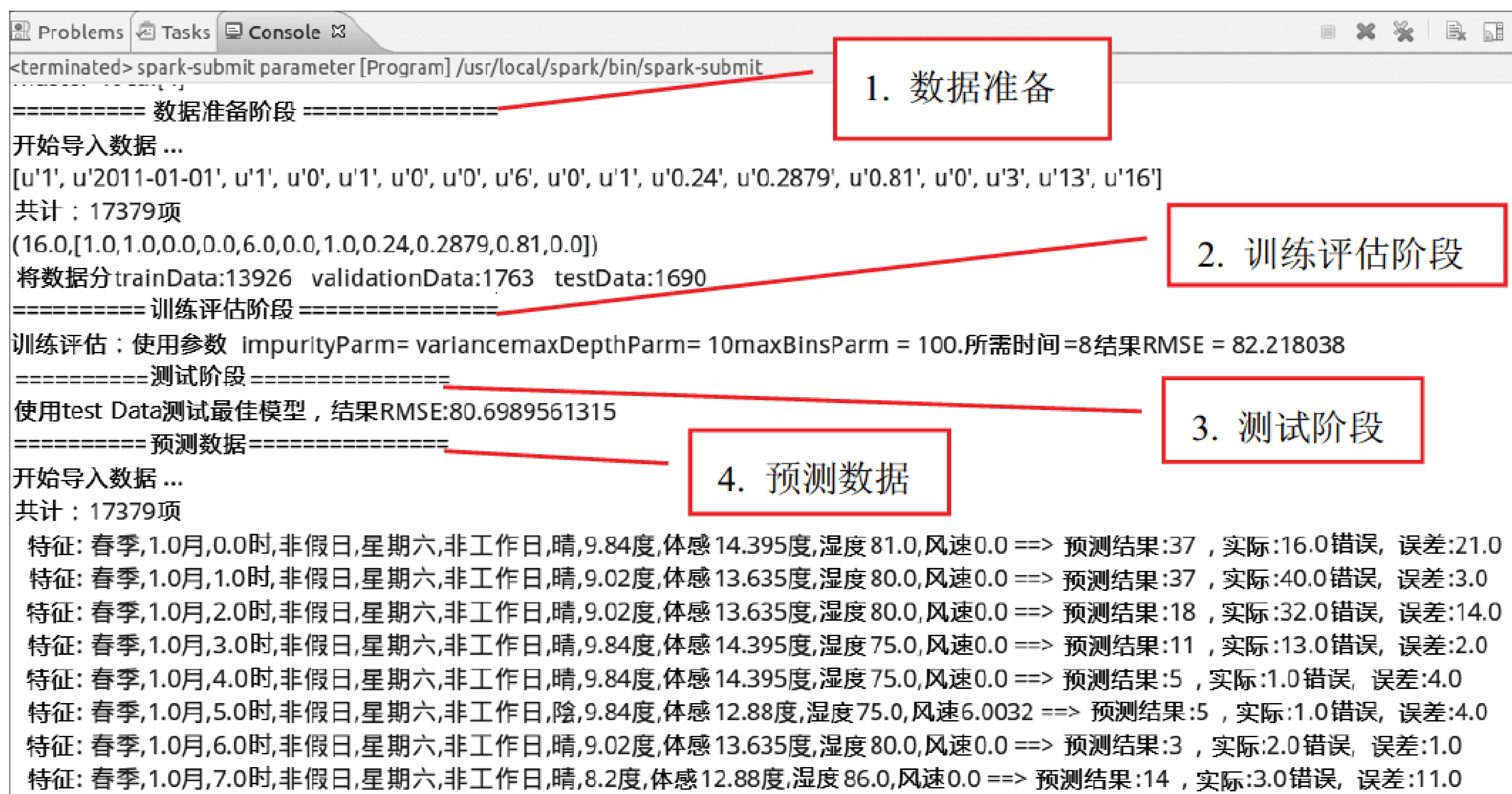


图 18-22　运行结果

（1）**数据准备阶段：** 显示导入项数以及 trainData、validationData、testData 项数。

（2）**训练评估阶段：** 进行最佳参数训练，impurity:variance，maxDepth:10，maxBins:100，结果 RMSE 大约为 82。

（3）**测试阶段：** 使用测试数据再测试模型，RMSE 大约为 80，与训练阶段差异不大，确认没有 overfitting 的问题。

（4）**预测阶段：** 使用 hour.csv 预测，显示在不同的天气条件下预测的值与实际的值并显示误差。

18.12 结论

本章介绍了 Spark MLlib 决策树回归分析，并且完成了数据准备、训练模型、存储模型、进行预测 Bike Sharing（共享单车）租借数量。后续第 19~22 章将介绍 DataFrame 与 SQL，以及 Spark ML pipeline: Dataframes-based 机器学习 API。

第 19 章

Python Spark SQL、DataFrame、RDD 数据统计与可视化

Spark 数据处理方式主要有 3 种：Spark SQL、DataFrame、RDD。本章我们将示范如何以这 3 种方式进行数据统计与数据可视化。

19.1 RDD、DataFrame、Spark SQL 比较

➢ RDD API

```
In [91]: userRDD.map(lambda x: (x[2],1)).reduceByKey(lambda x,y: x+y).collect()
Out[91]: [(u'M', 670), (u'F', 273)]
```

使用 RDD API 进行数据统计，主要是使用 map 配合 reduceByKey。RDD 的数据类型只有数据，没有定义 Schema，也就是说 RDD 未定义字段名及其数据类型，所以我们只能使用位置来指定每一个字段。例如，上述程序代码，我们要统计性别，必须以 map 方法配合 lambda 语句转换为 (x[2],1)，x[2] 就是性别字段，然后使用 reduceByKey 计算总和。因此我们必须要有 Map/Reduce 的概念，才能编写出下列程序代码。这通常是我们必须要有高级的程序设计能力才能够达到。不过，RDD 功能也最强，能完成所有 Spark 功能。

➢ DataFrame API

```
In [95]: user_df.select("gender").groupby("gender").count().show()

+------+-----+
|gender|count|
+------+-----+
|     F|  273|
|     M|  670|
+------+-----+
```

Spark DataFrame 被创建时必须定义 Schema，定义每一个字段名与数据类型，因而我们可以用字段名（例如 gender 性别）进行统计。DataFrame API 已经定义了很多类似 SQL 的方法，例如 select()、groupby()、count()，我们可以使用这些方法进行统计，比起 RDD 显然容易使用多了。只是使用 DataFrame API 必须要有基础的程序设计能力。

➢ Spark SQL

```
In [93]: sqlContext.sql("""
         SELECT gender ,count(*)  counts
         FROM   user_table
         GROUP BY gender """).show()

+------+------+
|gender|counts|
+------+------+
|     F|   273|
|     M|   670|
+------+------+
```

Spark SQL 是由 DataFrame 派生出来的，我们必须先创建 DataFrame，然后通过登录 Spark SQL temp table 就可以使用 Spark SQL 语句了。使用 Spark SQL 最简单，就是直接使用 SQL 语句，即使是非程序设计人员，只需要懂得 SQL 语句就可以使用。

19.2 创建 RDD、DataFrame 与 Spark SQL

为了让大家更容易理解 DataFrame 与 Spark SQL，我们使用 IPython Notebook 来示范，因为使用 IPython Notebook 具有互动性的好处，可以看到命令执行后的结果。以下示范在本地执行，读者也可以参考第 9.9 节的说明在不同的模式运行 IPython Notebook。读者可以参考附录 A 有关本书的范例程序下载与安装的说明去下载本章 IPython Notebook 范例文件进行练习。

19.2.1 在 local 模式运行 IPython Notebook

在“终端”程序中输入下列命令：

➢ **切换到 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **使用 IPython Notebook 界面在本地运行 pyspark**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888，显示 IPython Notebook 界面。

19.2.2 创建 RDD

步骤 01 配置文件读取路径

在 IPython Notebook 使用下列指令配置文件读取路径：

```
In [1]: global Path
        if sc.master[0:5]=="local" :
           Path="file:/home/hduser/pythonwork/PythonProject/"
        else:
           Path="hdfs://master:9000/user/hduser/"
```

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，代表当前是本地运行，读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，也就是在 cluster（群集）运行，就有可能是 YARN client 或 Spark stand alone，读取 HDFS 文件。

步骤 02 读取文本文件并且查看数据项数

在 IPython Notebook 使用下列指令读取文本文件并且查看数据笔数：

```
In [2]: RawUserRDD= sc.textFile(Path+"data/u.user")

In [3]: RawUserRDD.count()
Out[3]: 943
```

以上运行结果显示共有 943 项数据。

步骤03 查看前 5 项数据

在 IPython Notebook 使用指令查看前 5 项数据（见图 19-1）。

```
In [4]: RawUserRDD.take(5)

Out[4]: [u'1|24|M|technician|85711',
         u'2|53|F|other|94043',
         u'3|23|M|writer|32067',
         u'4|24|M|technician|43537',
         u'5|33|F|other|15213']
```


图 19-1 查看前 5 项数据

步骤04 按照"|"符号获取每一个字段

从之前的步骤可以看到，因为字段之间以"|"符号分隔，所以我们使用下列程序代码来获取每一个字段。

以下程序代码 RawUserRDD.map(lambda line:) 使用 map 处理每一项数据，用 lambda 语句创建匿名函数传入 line 参数。在匿名函数中，line.split("|") 按照"|"符号的分隔获取每一个字段。最后查看前 5 项数据。

```
In [7]: userRDD =RawUserRDD.map(lambda line: line.split("|"))
        userRDD .take(5)

Out[7]: [[u'1', u'24', u'M', u'technician', u'85711'],
         [u'2', u'53', u'F', u'other', u'94043'],
         [u'3', u'23', u'M', u'writer', u'32067'],
         [u'4', u'24', u'M', u'technician', u'43537'],
         [u'5', u'33', u'F', u'other', u'15213']]
```

19.2.3 创建 DataFrame

在上一小节我们创建了 userRDD。接下来，我们使用此 RDD 创建 DataFrame。

步骤01 创建 sqlContext

在 Spark 早期版本，spark context 是 Spark 的入口、sqlContext 是 SQL 的入口、HiveContext 是 hive 的入口。到了 Spark 2.0，我们只要使用 Spark Session 就可以同时具备 spark context、sqlContext、HiveContext 的功能。可以使用下列指令创建 sqlContext：

```
In [8]: sqlContext = SparkSession.builder.getOrCreate()
```



步骤 02　定义 Schema

然后，定义 DataFrames 的每一个字段名与数据类型，如图 19-2 所示。

```
In [9]: from pyspark.sql import Row
        user_Rows = userRDD.map(lambda p:
            Row(
              userid=int(p[0]),
              age=int(p[1]),
              gender=p[2],
              occupation=p[3],
              zipcode=p[4]
            )
        )
        user_Rows.take(5)

Out[9]: [Row(age=24, gender=u'M', occupation=u'technician', userid=1, zipcode=u'85711'),
         Row(age=53, gender=u'F', occupation=u'other', userid=2, zipcode=u'94043'),
         Row(age=23, gender=u'M', occupation=u'writer', userid=3, zipcode=u'32067'),
         Row(age=24, gender=u'M', occupation=u'technician', userid=4, zipcode=u'43537'),
         Row(age=33, gender=u'F', occupation=u'other', userid=5, zipcode=u'15213')]
```


图 19-2　定义 Schema

从以上运行结果我们可以看到每一项都是 Row 数据类型，并且已经定义了每一个字段的名称与数据类型。程序代码详细说明如表 19-1 所示。

表 19-1　程序代码说明

程序代码	说明
from pyspark.sql import Row	导入 Row 模块
user_Rows = userRDD.map(lambda p:	使用 userRDD.map 处理每一项数据，用 lambda 语句创建匿名函数，传入 p 参数作为每一项数
Row(userid=int(p[0]), age=int(p[1]), gender=p[2], occupation=p[3], zipcode=p[4])	在匿名函数中，使用 Row 传入每一个字段，创建 Row 数据类型。其中，userid 与 age 是数值，使用 int() 进行转换；其余是文字，不需要转换
user_Rows.take(5)	查看前 5 项 user_Row

步骤 03　创建 DataFrames

创建了 user_Rows 之后，使用 sqlContext. createDataFrame() 方法传入 user_Rows 数据，创建 DataFrame，然后使用 .printSchema() 方法查看 DataFrames 的 Schema。

```
In [10]: user_df = sqlContext.createDataFrame(user_Rows)
         user_df .printSchema()

root
 |-- age: long (nullable = true)
 |-- gender: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- userid: long (nullable = true)
 |-- zipcode: string (nullable = true)
```

以上运行结果显示 Schema，我们可以看到 age 与 userid 都是数值的 long 数据类型，其他的是 string 字符串数据类型。

步骤 04 查看 DataFrames 数据无法排列整齐

可以使用 show(5)查看前 5 项数据。有可能会看到图 19-3 所示的界面，数据无法排列整齐。这是由 IPython Note 默认字体的字宽不同导致的。

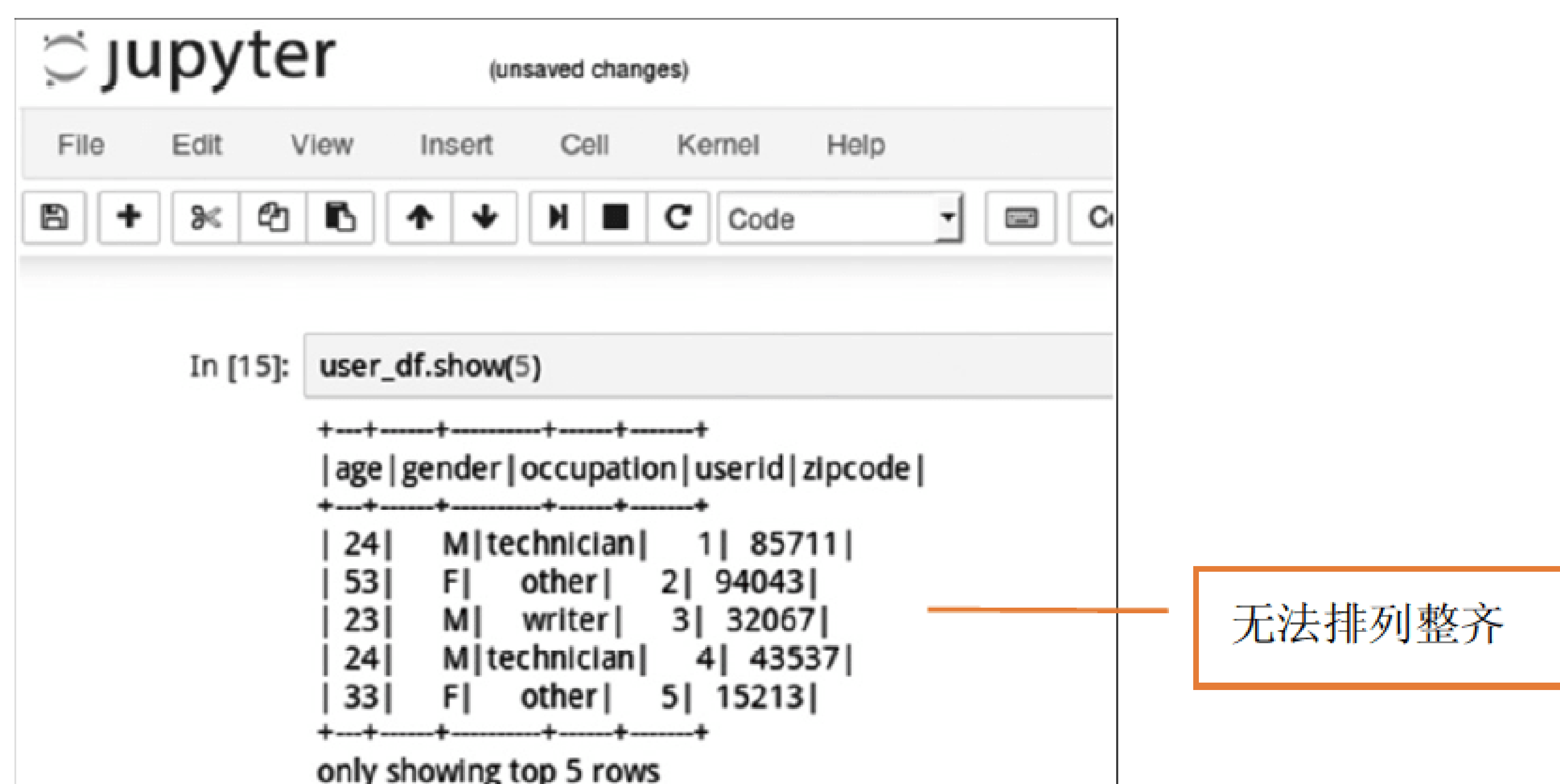


图 19-3 查看 DataFrames 数据无法排列整齐

19.2.4 设置 IPython Notebook 字体

因为 IPython Notebook 在浏览器中运行，所以我们可以通过修改 css 改变显示字体。

步骤 01 编辑 custom.css

在“终端”程序中输入下列命令：

➢ **使用 gedit 编辑 custom.css**

```
sudo gedit ~/anaconda2/lib/python2.7/site-packages/notebook/static/custom/custom.css
```

使用 gedit 打开 custom.css 后，输入下列 css 语句，设置字体为 Courier New、大小为 9pt。输入完成后存盘退出，如图 19-4 所示。

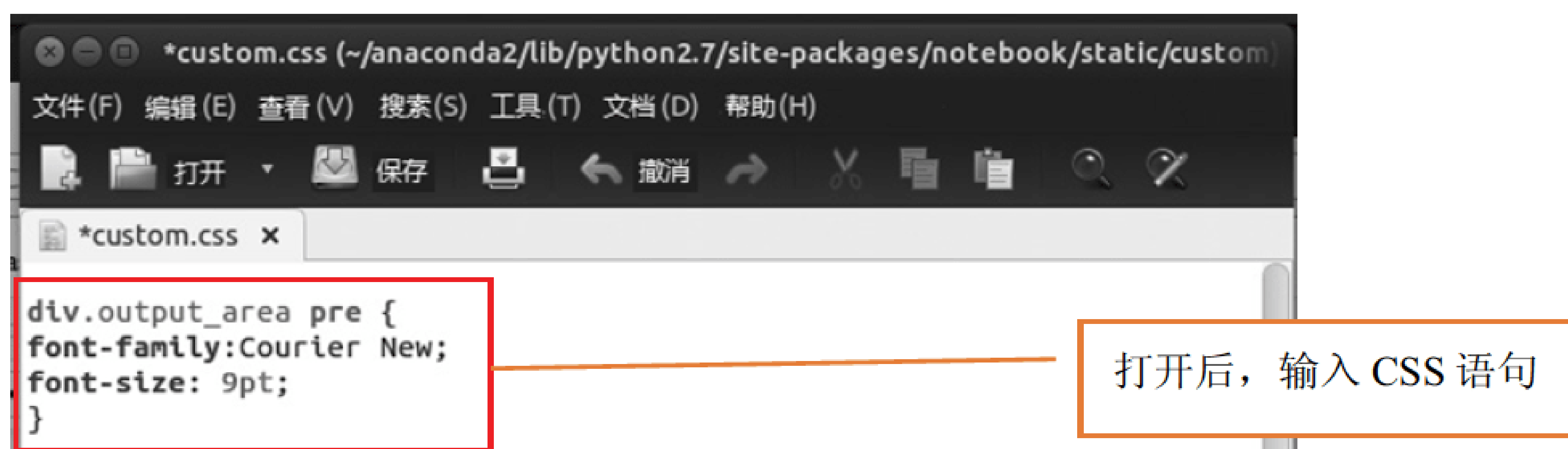


图 19-4 输入 CSS 语句

步骤 02　在浏览器中单击 Reload 图标

编辑完成后，在浏览器中单击 Reload 图标重新加载后，我们可以看到 DataFrame 已经排列整齐，如图 19-5 所示。

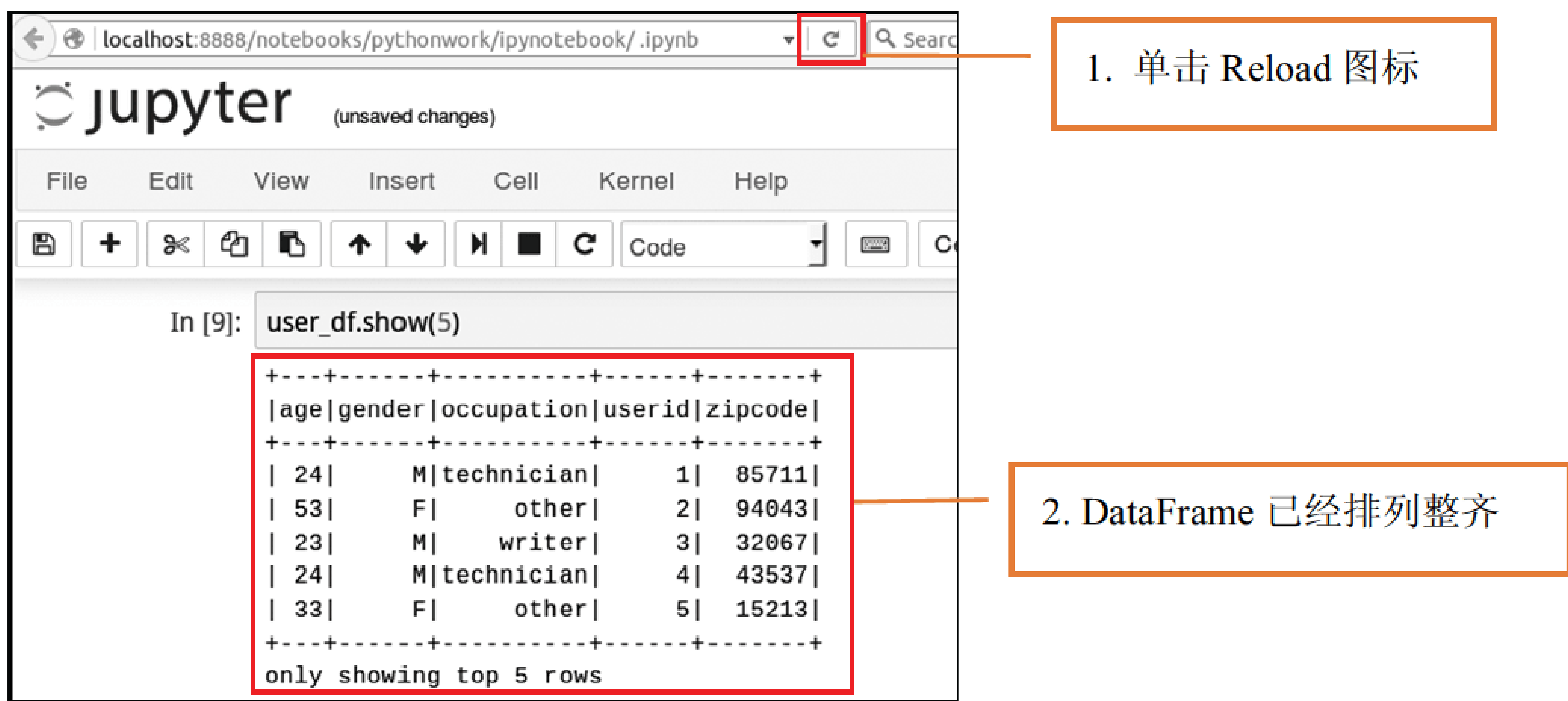


图 19-5　单击 Reload 图标

19.2.5　为 DataFrame 创建别名

有时 DataFrame 名称会很长，我们可以使用 .alias() 方法帮 DataFrame 创建别名。例如，user_df.alias("df")，后续我们就可以用这个别名执行命令了。

```
In [10]: df=user_df.alias("df")
         df.show(5)

+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 53|     F|     other|     2|  94043|
| 23|     M|    writer|     3|  32067|
| 24|     M|technician|     4|  43537|
| 33|     F|     other|     5|  15213|
+---+------+----------+------+-------+
only showing top 5 rows
```

以上运行结果与之前的步骤完全相同。

19.2.6　开始使用 Spark SQL

之前我们创建了 DataFrame，下面使用这个 DataFrame 登录 Spark SQL temp table，登录后就可以开始使用 Spark SQL 了。

步骤 01　登录临时表

user_df 使用 registerTempTable 方法传入数据表名称作为参数，登录临时表。

步骤02 使用 Spark SQL 查看项数

登录 Spark SQL temp table 后，我们就可以使用 Spark SQL 查看项数了。以下命令使用 sqlContext.sql() 输入 SQL 语句：使用 SELECT 关键词指定要显示 count(*) 数据项数，并使用 FROM 关键词指定要显示的数据表 user_table。最后使用 show() 显示结果。

```
In [14]: sqlContext.sql(" SELECT count(*) counts FROM user_table").show()

+------+
|counts|
+------+
|   943|
+------+
```

以上 SQL 语句执行后显示共有 943 项数据。

步骤03 多行输入 Spark SQL 语句

之前我们将 SQL 语句显示在同一行中，有时 SQL 语句很长，不容易阅读，也可以输入多行 Spark SQL 语句，只需要使用 3 个双引号 “"""” 引住 SQL 语句即可，如图 19-6 所示。

```
In [15]: sqlContext.sql("""
         SELECT count(*) counts
         FROM  user_table
         """).show()

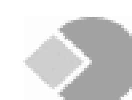
+------+
|counts|
+------+
|   943|
+------+
```


图 19-6 输入多行 Spark SQL 语句

运行后结果与上一步骤完全相同。

步骤04 使用 Spark SQL 查看数据

接下来，可以使用 SQL 指令查看 user_table 数据。运行后结果如图 19-7 所示。show() 方法默认显示前 20 项数据。



```
In [16]: sqlContext.sql(" SELECT *  FROM user_table ").show()

+---+------+-------------+------+-------+
|age|gender|   occupation|userid|zipcode|
+---+------+-------------+------+-------+
| 24|     M|   technician|     1|  85711|
| 53|     F|        other|     2|  94043|
| 23|     M|       writer|     3|  32067|
| 24|     M|   technician|     4|  43537|
| 33|     F|        other|     5|  15213|
| 42|     M|    executive|     6|  98101|
| 57|     M|administrator|     7|  91344|
| 36|     M|administrator|     8|  05201|
| 29|     M|      student|     9|  01002|
| 53|     M|       lawyer|    10|  90703|
| 39|     F|        other|    11|  30329|
| 28|     F|        other|    12|  06405|
| 47|     M|     educator|    13|  29206|
| 45|     M|    scientist|    14|  55106|
| 49|     F|     educator|    15|  97301|
| 21|     M|entertainment|    16|  10309|
| 30|     M|   programmer|    17|  06355|
| 35|     F|        other|    18|  37212|
| 40|     M|    librarian|    19|  02138|
| 42|     F|    homemaker|    20|  95660|
+---+------+-------------+------+-------+
only showing top 20 rows
```

show() 方法默认显示前 20 项数据

图 19-7　使用 Spark SQL 查看数据

步骤 05　使用 show() 方法指定要显示的数据项数

也可以使用 show() 方法传入参数来指定要显示的数据项数。运行后结果如图 19-8 所示，show(5) 方法显示前 5 项数据。

```
In [17]: sqlContext.sql(" SELECT *  FROM user_table").show(5)

+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 53|     F|     other|     2|  94043|
| 23|     M|    writer|     3|  32067|
| 24|     M|technician|     4|  43537|
| 33|     F|     other|     5|  15213|
+---+------+----------+------+-------+
only showing top 5 rows
```

显示前 5 项数据

图 19-8　使用 show()方法显示前 5 项数据

步骤 06　使用 Spark SQL LIMIT 指定要显示的项数

在步骤 5 中，SQL 语句产生的数据集是 943 项，但是这里只显示了 5 项。如果数据集很大，例如数十万项数据，可能会运行很久。此时我们可以直接使用 LIMIT 语句产生只有 5 项的数据集，即使数据量很大也不会运行很久，如图 19-9 所示。

```
In [91]: sqlContext.sql(" SELECT *  FROM user_table LIMIT 5").show()

+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 53|     F|     other|     2|  94043|
| 23|     M|    writer|     3|  32067|
| 24|     M|technician|     4|  43537|
| 33|     F|     other|     5|  15213|
+---+------+----------+------+-------+
```

LIMIT 5 显示前 5 项数据

图 19-9　使用 LZMIT 语句显示前 5 项数据

19.3 SELECT 显示部分字段

之前我们已经创建 RDD、DataFrames 与 Spark SQL，下面将分别使用 RDD、DataFrames 与 Spark SQL 语句来示范命令的使用。通过这种方式，我们可以比较不同语句的差异。下面首先示范显示部分字段。

19.3.1　使用 RDD 选取显示部分字段

当我们使用 RDD 选取显示部分字段时，因为没有 Schema，未定义字段名，所以只能指定位置。在这里我们要显示 userid、occupation、gender、age 字段，分别是第 0、3、2、1 字段，所以下列指令 userRDD.map(lambda x:) 处理每一项数据，并且使用 lambda 创建匿名函数，传入 x 作为参数，指定显示字段 x[0], x[3], x[2], x[1]，产生另一个 userRDDnew 的 RDD。最后使用 take(5) 显示前 5 项数据。

```
In [19]:  userRDDnew= userRDD.map(lambda x: (x[0],x[3],x[2] ,x[1]) )
          userRDDnew.take(5)

Out[19]: [(u'1', u'technician', u'M', u'24'),
          (u'2', u'other', u'F', u'53'),
          (u'3', u'writer', u'M', u'23'),
          (u'4', u'technician', u'M', u'24'),
          (u'5', u'other', u'F', u'33')]
```

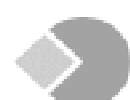
19.3.2　使用 DataFrames 选取显示字段

当使用 DataFrames 选取显示字段时，因为已经定义了 Schema，所以可以使用 select 方法输入字段名。下面使用 DataFrame 选取显示字段。有 4 种语句，执行结果相同。

（1）select 方法输入字段名字符串，选取显示字段。

```
In [20]: user_df.select("userid","occupation","gender","age").show(5)
```

（2）select 方法输入 [dataframe 名称].[字段名]，选取显示字段。

我们也可以使用 [dataframe 名称].[字段名]来指定显示数据。例如，下列指令使用 user_df.userid, user_df.occupation 指定要显示的字段，如图 19-10 所示。

```
In [20]: user_df.select( user_df.userid,user_df.occupation,user_df.gender,user_df.age ).show(5)
```

Dataframe 名称

图 19-10　使用 [dataframe 名称].[字段名]的方法选取显示字段

（3）select 方法输入 [dataframe 别名].[字段名]，选取显示字段。

之前的方式使用 [dataframe 名称].[字段名]选取显示字段，有时会让程序代码冗长，不易阅读。我们也可以使用 [dataframe 别名].[字段名]来选取显示字段。在第 19.2.5 节中，我们已经创建了别名 df，所以可以使用 df.userid, df.occupation, df.gender,df.age 来指定要显示的字段，如图 19-11 所示。

```
In [22]: df.select(df.userid,df.occupation,df.gender,df.age ).show(5)
```

Dataframe 别名

图 19-11　使用别名来选取显示字段

（4）select 方法输入中括号“[”和“]”，选取显示字段。

```
In [24]: df[df['userid'],df['occupation'],df['gender'],df['age'] ].show(5)
```

以上 4 种 DataFrames 选取显示字段的语句，运行结果完全相同，如图 19-12 所示。

```
+------+----------+------+---+
|userid|occupation|gender|age|
+------+----------+------+---+
|     1|technician|     M| 24|
|     2|     other|     F| 53|
|     3|    writer|     M| 23|
|     4|technician|     M| 24|
|     5|     other|     F| 33|
+------+----------+------+---+
only showing top 5 rows
```

图 19-12　最终的运行结果

19.3.3　使用 Spark SQL 选取显示字段

在 SQL 语句中，我们可以使用 SELECT 关键词来指定要显示的字段：

```
In [39]: sqlContext.sql(" SELECT userid,occupation,gender,age  FROM user_table").show(5)

+------+----------+------+---+
|userid|occupation|gender|age|
+------+----------+------+---+
|     1|technician|     M| 24|
|     2|     other|     F| 53|
|     3|    writer|     M| 23|
|     4|technician|     M| 24|
|     5|     other|     F| 33|
+------+----------+------+---+
only showing top 5 rows
```

19.4 增加计算字段

当我们显示数据时，某些字段必须经过计算，例如用户的年龄字段。我们想要知道用户的出生年份时可以使用年份，就用年份（比如 2016）减年龄，可以大概知道用户的出生年份，此时就需要使用计算字段了。

19.4.1 使用 RDD 增加计算字段

以下命令与之前 RDD 显示字段类似，只是我们加上了 2016-int(x[1])，用于计算出生年份，并显示出生年份，如图 19-13 所示。

```
In [26]: userRDDnew= userRDD.map(lambda x: (x[0],x[3],x[2] , x[1],2016-int(x[1])) )
         userRDDnew.take(5)

Out[26]: [(u'1', u'technician', u'M', u'24', 1992),
          (u'2', u'other', u'F', u'53', 1963),
          (u'3', u'writer', u'M', u'23', 1993),
          (u'4', u'technician', u'M', u'24', 1992),
          (u'5', u'other', u'F', u'33', 1983)]
```


图 19-13 使用 RDD 增加计算字段

19.4.2 使用 DataFrames 增加计算字段

使用 DataFrames 增加计算字段比 RDD 简单得多，因为 DataFrames 具有 Schema 信息，所以我们可以直接输入字段名。

步骤 01 使用 DataFrames 增加计算字段

下列指令（见图 19-14）加入 2016-df.age，增加了计算字段来计算出生年份。

```
In [27]: df.select("userid","occupation","gender","age",2016-df.age).show(5)

+------+----------+------+---+------------+
|userid|occupation|gender|age|(2016 - age)|
+------+----------+------+---+------------+
|     1|technician|     M| 24|        1992|
|     2|     other|     F| 53|        1963|
|     3|    writer|     M| 23|        1993|
|     4|technician|     M| 24|        1992|
|     5|     other|     F| 33|        1983|
+------+----------+------+---+------------+
only showing top 5 rows
```

计算字段是 2016-df.age

字段名称是“(2016-age)”

图 19-14　使用 DataFrames 增加计算字段

步骤 02 为计算字段取一个别名

以上步骤，我们看到的字段名是“(2016 - age)”，我们希望能为计算字段取一个别名，使用“(2016-df.age).alias("birthyear")”将此字段的别名设置为“birthyear”，如图 19-15 所示。

```
In [29]: df.select("userid","occupation","gender","age",(2016-df.age).alias("birthyear ")).show(5)

+------+----------+------+---+----------+
|userid|occupation|gender|age|birthyear |
+------+----------+------+---+----------+
|     1|technician|     M| 24|      1992|
|     2|     other|     F| 53|      1963|
|     3|    writer|     M| 23|      1993|
|     4|technician|     M| 24|      1992|
|     5|     other|     F| 33|      1983|
+------+----------+------+---+----------+
only showing top 5 rows
```

为计算字段设置别名

字段名是“birthyear”

图 19-15　为计算字段设置别名

19.4.3　使用 Spark SQL 增加计算字段

使用 Spark SQL 语句增加计算字段也很简单。如图 19-16 所示，这里增加了“2016-age birthyear”语句来计算出生年份，并指定字段名为“birthyear”。

```
In [29]: sqlContext.sql("""
         SELECT userid,occupation,gender,age,2016-age birthyear
         FROM user_table""").show(5)

+------+----------+------+---+---------+
|userid|occupation|gender|age|birthyear|
+------+----------+------+---+---------+
|     1|technician|     M| 24|     1992|
|     2|     other|     F| 53|     1963|
|     3|    writer|     M| 23|     1993|
|     4|technician|     M| 24|     1992|
|     5|     other|     F| 33|     1983|
+------+----------+------+---+---------+
only showing top 5 rows
```

1. 为计算字段设置名称 birthyear

2. 已设置字段名是“birthyear”

图 19-16

19.5 筛选数据

接下来介绍如何筛选我们所需要的数据。

19.5.1 使用 RDD 筛选数据

在 RDD 中我们使用 filter 方法筛选每一项数据，配合 lambda 语句创建匿名函数传入参数 r。在匿名函数中，使用“r[3]=='technician' and r[2]=='M' and r[1]=='24'”语句（见图 19-17），意义是：r[3](职业 occupation)='technician'，而且 r[2](性别 gender)='M'(男性)，而且 r[1](年龄 age)=24 岁。

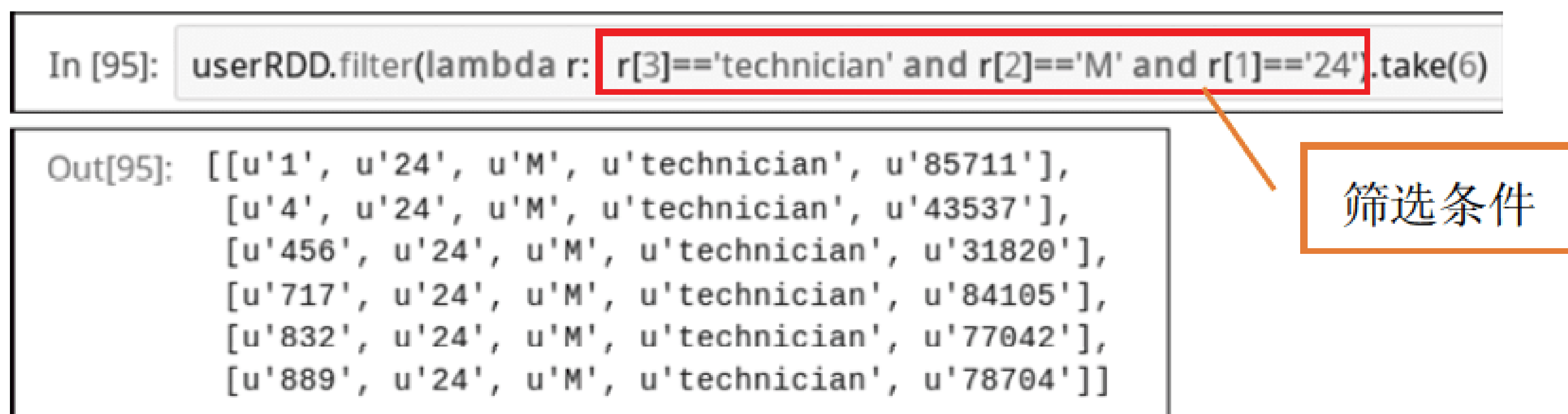


图 19-17 使用 RDD 筛选数据

19.5.2 使用 DataFrames 筛选数据

接下来，使用 DataFrames 筛选数据。有下列 4 种语句，执行结果相同。

1. 使用多个 filter 筛选数据

以下指令使用多个 filter 筛选数据，多个 filter 相当于 and（“且”）。

```
In [31]: user_df.filter("occupation='technician' ").filter("gender='M' ").filter("age=24").show()
```

2. 使用单个 filter 筛选数据

使用单个 filter 输入语句筛选数据，在这里我们使用 and 连接条件。使用这种语句更有弹性，因为我们除了使用 and 外，还可以使用 or 或 not 进行筛选。

3. 使用 [dataframe 名称].[字段名]指定筛选条件

另外，还可以使用[dataframe 名称].[字段名]来指定筛选条件。使用这种语句时，请注意下列两种方式：

- 必须使用“&”，而不能使用“and”。
- 必须使用“==”，而不能使用“=”。

```
In [33]: df.filter((df.occupation=='technician' ) & (df.gender=='M' ) & (df.age==24)).show()
```

4. 使用中括号 [ ] 指定筛选条件

我们还可以使用中括号 [ ] 指定筛选条件，如下列指令：

```
In [92]: df.filter((df['occupation']=='technician' ) & (df['gender']=='M' ) & (df['age']==24)).show()
```

以上 4 种 DataFrames 命令运行的结果完全相同，如图 19-18 所示。

```
+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 24|     M|technician|     4|  43537|
| 24|     M|technician|   456|  31820|
| 24|     M|technician|   717|  84105|
| 24|     M|technician|   832|  77042|
| 24|     M|technician|   889|  78704|
+---+------+----------+------+-------+
```

图 19-18　使用 DataFrames 筛选数据

19.5.3　使用 Spark SQL 筛选数据

使用 Spark SQL 筛选数据很简单，只需要使用 where 语句输入筛选条件即可：

```
In [36]: sqlContext.sql(
         '''SELECT *
         FROM user_table
         where occupation='technician' and  gender='M' and age=24''').show(5)

+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 24|     M|technician|     4|  43537|
| 24|     M|technician|   456|  31820|
| 24|     M|technician|   717|  84105|
| 24|     M|technician|   832|  77042|
+---+------+----------+------+-------+
only showing top 5 rows
```

19.6　按单个字段给数据排序

19.6.1　RDD 按单个字段给数据排序

在 RDD 中，可以使用 takeOrdered(num, key=None) 方法对数据进行排序：

- 第 1 个参数 num：要显示的项数。
- 第 2 个参数 key：使用 lambda 语句设置要排序的字段。

➢ 使用 RDD 按升序给数据排序

以下命令用于设置 key=lambda x:int(x[1])，也就是按 x[1]年龄从小到大排序（见图 19-19）。

```
In [36]: userRDD.takeOrdered(5, key = lambda x: int(x[1]))
Out[36]: [[u'30', u'7', u'M', u'student', u'55436'],
          [u'471', u'10', u'M', u'student', u'77459'],
          [u'289', u'11', u'M', u'none', u'94619'],
          [u'142', u'13', u'M', u'other', u'48118'],
          [u'609', u'13', u'F', u'student', u'55106']]
```

设置按年龄升序排序

按照年龄升序排序

图 19-19 使用 RDD 按升序排序

➢ 使用 RDD 按降序给数据排序

以下命令用于设置 key = lambda -1*x: int(x[1])，也就是按年龄从大到小排序（见图 19-20）。

```
In [43]: userRDD.takeOrdered(5, key = lambda x: -1*int(x[1]))
Out[43]: [[u'481', u'73', u'M', u'retired', u'37771'],
          [u'767', u'70', u'M', u'engineer', u'00000'],
          [u'803', u'70', u'M', u'administrator', u'78212'],
          [u'860', u'70', u'F', u'retired', u'48322'],
          [u'559', u'69', u'M', u'executive', u'10022']]
```

设置按年龄降序排序

按照年龄降序排序

图 19-20 使用 RDD 按降序排序

19.6.2 使用 Spark SQL 排序

➢ Spark SQL 按升序给数据排序

在下列 SQL 语句中，我们使用 ORDER BY 关键词加上字段名来指定排序字段，默认为按升序排序，如图 19-21 所示。

```
In [38]: sqlContext.sql("""
         SELECT userid,occupation,gender,age
         FROM user_table
         ORDER BY age""").show(5)

+------+----------+------+---+
|userid|occupation|gender|age|
+------+----------+------+---+
|    30|   student|     M|  7|
|   471|   student|     M| 10|
|   289|      none|     M| 11|
|   142|     other|     M| 13|
|   609|   student|     F| 13|
+------+----------+------+---+
only showing top 5 rows
```

设置按年龄升序排序

按照年龄升序排序

图 19-21 使用 Spark SQL 按升序排序



➢ **Spark SQL 按降序给数据排序**

在下列 SQL 语句中，与之前步骤类似，唯一不同的地方是我们加入了 DESC 关键词，设置按降序给数据排序，如图 19-22 所示。

```
In [39]: sqlContext.sql("""
         SELECT userid,occupation,gender,age
         FROM user_table
         ORDER BY age DESC""").show(5)

+------+-------------+------+---+
|userid|   occupation|gender|age|
+------+-------------+------+---+
|   481|      retired|     M| 73|
|   767|     engineer|     M| 70|
|   860|      retired|     F| 70|
|   803|administrator|     M| 70|
|   559|    executive|     M| 69|
+------+-------------+------+---+
only showing top 5 rows
```

设置按年龄降序排序

按照年龄降序排序

图 19-22　使用 Spark SQL 按降序排序

19.6.3　使用 DataFrames 按升序给数据排序

使用 DataFrames 按升序给数据排序。默认设置就是升序。

➢ **使用 .orderBy("age") 按升序给数据排序**

我们可以使用 .orderBy("age") 输入要排序的字段名来进行排序。默认设置是升序，所以我们不需要注明 ascending。

```
In [46]: user_df.select("userid","occupation","gender","age").orderBy("age").show(5)
```

➢ **使用 orderBy(df.age) 按升序给数据排序**

我们也可以使用 orderBy(df.age) 指定排序字段，默认为升序。

```
In [77]: df.select("userid","occupation","gender","age").orderBy(df.age).show(5)
```

以上两种 DataFrames 语句的运行结果完全相同，如图 19-23 所示。

```
+------+----------+------+---+
|userid|occupation|gender|age|
+------+----------+------+---+
|    30|   student|     M|  7|
|   471|   student|     M| 10|
|   289|      none|     M| 11|
|   142|     other|     M| 13|
|   609|   student|     F| 13|
+------+----------+------+---+
only showing top 5 rows
```

按照年龄升序排序

图 19-23　使用 DataFrames 按升序排列

19.6.4 使用 DataFrames 按降序给数据排序

使用 DataFrames 按降序给数据排序，因为降序不是默认的设置，所以必须特别指明。

➢ **使用 .orderBy("age",ascending=0)按降序给数据排序**

```
In [75]: df.select("userid","occupation","gender","age").orderBy("age",ascending=0 ).show(5)
```

➢ **使用 df.age.desc()指定排序字段（.desc() 代表降序排序）**

```
In [50]: df.select("userid","occupation","gender","age").orderBy(df.age.desc()).show(5)
```

以上两种 DataFrames 语句的运行结果完全相同，如图 19-24 所示。

```
+------+-------------+------+---+
|userid|   occupation|gender|age|
+------+-------------+------+---+
|   481|      retired|     M| 73|
|   767|     engineer|     M| 70|
|   860|      retired|     F| 70|
|   803|administrator|     M| 70|
|   559|    executive|     M| 69|
+------+-------------+------+---+
only showing top 5 rows
```

按照年龄降序排序

图 19-24 使用 DataFrames 按降序排列

19.7 按多个字段给数据排序

接下来，我们将示范按多个字段给数据排序。例如，在下列范例中，我们将按年龄 age 降序排序，按性别 gender 升序排序。

19.7.1 RDD 按多个字段给数据排序

下列指令设置排序 key 为 lambda x: (-int(x[1]), x[2])，以 -int(x[1]) 设置按年龄 age 降序排序，并且以 x[2] 设置按性别 gender 升序排序，如图 19-25 所示。

```
In [43]: userRDD.takeOrdered(5, key = lambda x: (-int(x[1]), x[2] ))
Out[43]: [[u'481', u'73', u'M', u'retired', u'37771'],
          [u'860', u'70', u'F', u'retired', u'48322'],
          [u'767', u'70', u'M', u'engineer', u'00000'],
          [u'803', u'70', u'M', u'administrator', u'78212'],
          [u'559', u'69', u'M', u'executive', u'10022']]
```

先以年龄降序排序，如果年龄都是 70 时，再以性别升序排序

图 19-25 RDD 按多个字段给数据排序

19.7.2　Spark SQL 按多个字段给数据排序

在以下 Spark SQL 语句中，我们使用 ORDER BY age DESC 设置按年龄进行降序排序，而性别 gender 按默认的升序排序，如图 19-26 所示。

```
In [51]: sqlContext.sql("""
         SELECT userid, age, gender,occupation,zipcode
         FROM user_table
         ORDER BY age DESC,gender """).show(5)

+------+---+------+-------------+-------+
|userid|age|gender|   occupation|zipcode|
+------+---+------+-------------+-------+
|   481| 73|     M|      retired|  37771|
|   860| 70|     F|      retired|  48322|
|   767| 70|     M|     engineer|  00000|
|   803| 70|     M|administrator|  78212|
|   559| 69|     M|    executive|  10022|
+------+---+------+-------------+-------+
only showing top 5 rows
```

先以年龄降序排序，年龄都是 70 时，再以性别升序排序

图 19-26　Spark SQL 按多个字段给数据排序

19.7.3　DataFrames 按多个字段给数据排序

➢ 使用 orderBy(["age","gender"],ascending=[0,1] 按多个字段给数据排序

在以下 DataFrames 语句中，orderBy 方法设置说明如下：

- 第一个参数：设置要排序的字段：["age","gender"]。
- 第二个参数：设置排序字段的升序 / 降序：[0,1]。第一个字段 age 设置为 0，表示是升序；第二个字段 gender 设置为 1，表示是降序。

```
In [52]: df.orderBy(["age","gender"],ascending=[0,1] ).show(5)
```

➢ 使用 .orderBy(df.age.desc(),df.gender) 按多个字段给数据排序

我们也可以使用 .orderBy(df.age.desc(),df.gender) 指定第一个字段 age 按降序排序，第二个字段 gender 按升序排列。

```
In [53]: df.orderBy(df.age.desc(),df.gender ).show(5)
```

以上两种 DataFrames 语句的运行结果完全相同，如图 19-27 所示。

```
+---+------+-------------+------+-------+
|age|gender|   occupation|userid|zipcode|
+---+------+-------------+------+-------+
| 73|     M|      retired|   481|  37771|
| 70|     F|      retired|   860|  48322|
| 70|     M|     engineer|   767|  00000|
| 70|     M|administrator|   803|  78212|
| 69|     M|    executive|   559|  10022|
+---+------+-------------+------+-------+
only showing top 5 rows
```

先以年龄降序排序，年龄都是 70 时，再以性别升序排序

图 19-27　DataFrames 按多个字段给数据排序

19.8 显示不重复的数据

本节将介绍如何显示不重复的数据。

19.8.1 RDD 显示不重复的数据

步骤01 显示性别字段不重复的数据

使用下列指令显示 userRDD 性别字段不重复的数据：

```
In [48]: userRDD.map( lambda x:x[2] ).distinct().collect()
Out[48]: [u'M', u'F']
```

以上运行结果只显示 'M' 与 'F' 这两项。程序代码说明如表 19-2 所示。

表 19-2 程序代码说明

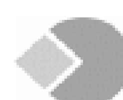
程序代码	说明
userRDD.map(lambda x: (x[2]))	使用 map 处理每一项数据，以 lambda 语句创建匿名函数传入 x 参数。在匿名函数中用 x[2] 筛选出只有性别数据的数据集
.distinct()	筛选出不重复的数据
collect()	转换为 List

步骤02 显示年龄 + 性别字段不重复的数据

我们也可以使用多个字段来显示不重复的数据，例如下面的范例显示年龄 + 性别字段不重复的数据。下列指令在匿名函数中使用 (x[1],x[2]) 来指定年龄 + 性别字段不重复的数据。

```
In [55]: userRDD.map( lambda x:(x[1],x[2]) ).distinct().take(20)
Out[55]: [(u'30', u'F'),
          (u'48', u'F'),
          (u'35', u'F'),
          (u'35', u'M'),
          (u'43', u'F'),
          (u'18', u'M'),
```

以上运行结果会显示年龄 + 性别字段不重复的数据，实际的项数很多，限于版面，这里只列出几项数据。

19.8.2 Spark SQL 显示不重复的数据

步骤 01 显示性别字段不重复的数据

在 Spark SQL 中，在字段名之前使用 distinct 关键词来设置显示不重复的数据。例如，在下列程序代码中，distinct gender 即为设置显示性别字段不重复的数据。

```
In [56]: sqlContext.sql(" SELECT distinct gender FROM user_table").show()

+------+
|gender|
+------+
|     F|
|     M|
+------+
```

步骤 02 显示年龄 + 性别字段不重复的数据

下列 Spark SQL 语句使用 distinct age, gender 来设置显示年龄 + 性别字段不重复的数据。

```
In [57]: sqlContext.sql(" SELECT distinct age,gender  FROM user_table").show()

+---+------+
|age|gender|
+---+------+
| 39|     F|
| 48|     M|
| 26|     M|
| 28|     M|
| 54|     M|
| 60|     M|
| 50|     M|
| 53|     F|
```

19.8.3 Dataframes 显示不重复的数据

步骤 01 显示性别字段不重复的数据

下列指令使用 DataFrames 显示不重复的数据，select("gender")选择要显示的字段，.distinct() 去除重复的数据。

```
In [58]: user_df.select("gender").distinct().show()

+------+
|gender|
+------+
|     F|
|     M|
+------+
```

步骤 02 显示年龄 + 性别字段不重复的数据

下列指令使用 DataFrames 显示年龄 + 性别字段不重复的数据，select("age","gender") 选择要显示的字段，.distinct() 去除重复的数据。

```
In [59]: user_df.select("age","gender").distinct().show()

+---+------+
|age|gender|
+---+------+
| 39|     F|
| 48|     M|
| 26|     M|
| 28|     M|
| 54|     M|
| 60|     M|
```

19.9 分组统计数据

在实际运用中，我们常常需要按照一个或多个字段进行分组统计。

19.9.1 RDD 分组统计数据

步骤 01 按照性别统计数据

在 RDD 中要进行数据的分组统计，必须使用 map/reduce。例如，按照性别统计数据。

```
In [60]: userRDD.map(lambda x: (x[2],1)) \
                .reduceByKey(lambda x,y: x+y).collect()

Out[60]: [(u'M', 670), (u'F', 273)]
```

程序代码说明如表 19-3 所示。

表 19-3 程序代码说明

程序代码	说明
userRDD.map(lambda x: (x[2],1))	userRDD 每一项数据通过 map 对应产生（性别,1）的数据集，例如（'M', 1）或（'F', 1）
.reduceByKey(lambda x,y: x+y)	将数据集分别按照性别计算总和来进行统计

步骤 02 按照性别、职业来统计数据

下列程序代码按照性别、职业来统计数据。

```
In [61]: userRDD.map(lambda x: ((x[2],x[3]),1)).reduceByKey(lambda x,y: x+y).collect()

Out[61]: [((u'F', u'healthcare'), 11),
          ((u'F', u'librarian'), 29),
          ((u'F', u'student'), 60),
          ((u'F', u'engineer'), 2),
          ((u'M', u'executive'), 29),
          ((u'M', u'healthcare'), 5),
          ((u'M', u'marketing'), 16),
          ((u'M', u'lawyer'), 10),
```

以上程序代码说明如表 19-4 所示。

表 19-4　程序代码说明

程序代码	说明
userRDD. map(lambda x: ((x[2], x[3]), 1))	userRDD 每一项数据通过 map 对应产生 ((性别, 职 业), 1) 的数据集，例如 (('F', 'healthcare'), 1) 或 (('M', 'executive'),1)
.reduceByKey(lambda x, y: x+y)	将数据集分别按照（性别，职业）计算总和来进行统计

19.9.2　Spark SQL 分组统计数据

当使用 Spark SQL 统计数据时，相对于 RDD 简单得多。

步骤 01 按照性别统计数据

下列 Spark SQL 语句按照性别统计数据：

```
In [62]: sqlContext.sql("""
         SELECT gender ,count(*)  counts
         FROM   user_table
         GROUP BY gender""").show()

         +------+------+
         |gender|counts|
         +------+------+
         |     F|   273|
         |     M|   670|
         +------+------+
```

以上 SQL 语句说明如表 19-5 所示。

表 19-5　SQL 语句说明

程序代码	说明
SELECT gender , count(*) counts	设置显示性别，count(*) 计算总和，将 count(*) 别名设置为 counts
FROM user_table	读取 User Table
GROUP BY gender	设置性别字段分组统计

步骤 02 按照性别、职业统计数据

下列 Spark SQL 语句按照性别、职业统计数据：

```
In [63]: sqlContext.sql("""
SELECT gender,occupation,count(*) counts
FROM   user_table
GROUP BY gender,occupation
""").show(100)

+------+-------------+------+
|gender|   occupation|counts|
+------+-------------+------+
|     M|    executive|    29|
|     M|     educator|    69|
|     F|         none|     4|
|     F|entertainment|     2|
|     F|      retired|     1|
```

以上SQL 语句说明如表 19-6 所示。

表 19-6　SQL 语句说明

程序代码	说明
SELECT gender,occupation, count(*) counts	设置显示性别、职业字段，count(*) 计算总和，并将 count(*) 别名设置为 counts
FROM user_table	读取 User Table
GROUP BY gender,occupation	设置性别、职业字段分组统计

19.9.3　Dataframes 分组统计数据

步骤 01 Dataframes 按照性别分组统计数据

以下 DataFrames 程序代码按照性别统计数据：

```
In [61]: user_df.select("gender").groupby("gender").count().show()

+------+-----+
|gender|count|
+------+-----+
|     F|  273|
|     M|  670|
+------+-----+
```

以上程序代码说明如表 19-7 所示。

表 19-7　程序代码说明

程序代码	说明
user_df.select("gender")	user_df. 使用 select("gender") 设置性别字段分组统计
.groupby("gender")	设置按照性别 gender 进行分组统计
.count()	进行分组统计
.show()	显示数据

步骤 02 Dataframes 按照性别、职业统计数据

以下 DataFrames 程序代码按照性别、职业统计数据：

```
In [65]: user_df.select("gender","occupation").             \
                 groupby("gender","occupation").             \
                 count().                                    \
                 orderBy("gender","occupation").             \
                 show(100)

+------+-------------+-----+
|gender|   occupation|count|
+------+-------------+-----+
|     F|administrator|   36|
|     F|       artist|   13|
|     F|     educator|   26|
|     F|     engineer|    2|
|     F|entertainment|    2|
|     F|    executive|    3|
```

以上程序代码分为多行，所以用“\”符号连接命令，说明如表 19-8 所示。

表 19-8　程序代码说明

程序代码	说明
user_df.	使用 user_df DataFrames 进行统计
select("gender","occupation").	设置性别、职业字段分组统计
groupby("gender","occupation").	设置按照性别、职业进行分组统计
count().	进行计算总和统计
orderBy("gender","occupation").	按照 "gender","occupation" 排序
show()	显示数据

步骤 03　以 crosstab 按照性别、职业统计数据

执行之前的指令会让数据显示很长，不易阅读，我们可以使用 crosstab 按照性别、职业统计数据。

```
In [66]: user_df.stat.crosstab("occupation","gender" ).show(30)

+-----------------+---+---+
|occupation_gender|  F|  M|
+-----------------+---+---+
|        scientist|  3| 28|
|          student| 60|136|
|           writer| 19| 26|
|         salesman|  3|  9|
|          retired|  1| 13|
|    administrator| 36| 43|
|       programmer|  6| 60|
|           doctor|  0|  7|
|        homemaker|  6|  1|
|        executive|  3| 29|
|         engineer|  2| 65|
|    entertainment|  2| 16|
|        marketing| 10| 16|
|       technician|  1| 26|
|           artist| 13| 15|
|        librarian| 29| 22|
|           lawyer|  2| 10|
|         educator| 26| 69|
|       healthcare| 11|  5|
|             none|  4|  5|
|            other| 36| 69|
+-----------------+---+---+
```

从以上运行结果可以发现这种方式更易阅读。

19.10 Join 联接数据

如图 19-28 所示，user_table 有 zipcode 字段，但是该字段是数字，对于用户毫无意义。

```
+---+------+----------+------+-------+
|age|gender|occupation|userid|zipcode|
+---+------+----------+------+-------+
| 24|     M|technician|     1|  85711|
| 53|     F|     other|     2|  94043|
| 23|     M|    writer|     3|  32067|
| 24|     M|technician|     4|  43537|
| 33|     F|     other|     5|  15213|
+---+------+----------+------+-------+
only showing top 5 rows
```

只有 zipcode 对用户无意义

图 19-28　zipcode 字段对用户无意义

如果希望知道用户来自于哪一个州，那么我们必须先下载 ZipCode 数据表，然后联接 ZipCode 数据表。

19.10.1 创建 ZipCode

步骤 01 下载 ZipCode 数据集

首先，以下列指令下载 ZipCode 数据集：

```
cd ~/pythonwork/PythonProject/data
wget http://federalgovernmentzipcodes.us/free-zipcode-database-Primary.csv
```

运行后结果如图 19-29 所示。

```
hduser@master: ~/pythonwork/PythonProject/data
hduser@master:~$ cd ~/pythonwork/PythonProject/data
hduser@master:~/pythonwork/PythonProject/data$ wget http://federalgovernmentzipc
odes.us/free-zipcode-database-Primary.csv
--2016-08-13 17:16:10--  http://federalgovernmentzipcodes.us/free-zipcode-databa
se-Primary.csv
正在解析主机 federalgovernmentzipcodes.us (federalgovernmentzipcodes.us)... 54.1
48.51.94
正在连接 federalgovernmentzipcodes.us (federalgovernmentzipcodes.us)|54.148.51.9
4|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 4318308 (4.1M) [application/octet-stream]
Saving to: ‘free-zipcode-database-Primary.csv’

100%[======================================>] 4,318,308   1.13MB/s   in 3.8s

2016-08-13 17:16:14 (1.08 MB/s) - ‘free-zipcode-database-Primary.csv’ saved [431
8308/4318308]
```

图 19-29　下载 ZipCode 数据集

步骤 02 复制到 HDFS

➢ **复制到 HDFS data 目录**

```
hadoop fs -copyFromLocal free-zipcode-database-Primary.csv /user/hduser/data
```

➢ HDFS data 目录

```
hadoop fs -ls /user/hduser/data/free-zipcode-database-Primary.csv
```

```
hduser@master: ~/pythonwork/PythonProject/data
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -copyFromLocal free-zip
code-database-Primary.csv /user/hduser/data
hduser@master:~/pythonwork/PythonProject/data$ hadoop fs -ls /user/hduser/data/f
ree-zipcode-database-Primary.csv
-rw-r--r--   3 hduser supergroup    4318308 2016-08-13 17:27 /user/hduser/data/f
ree-zipcode-database-Primary.csv
```

步骤 03 读取并查看数据

利用 sc.textFile 读取 csv 文件，并使用 take(2) 查看前两项数据，即字段名称与数据，如图 19-30 所示。

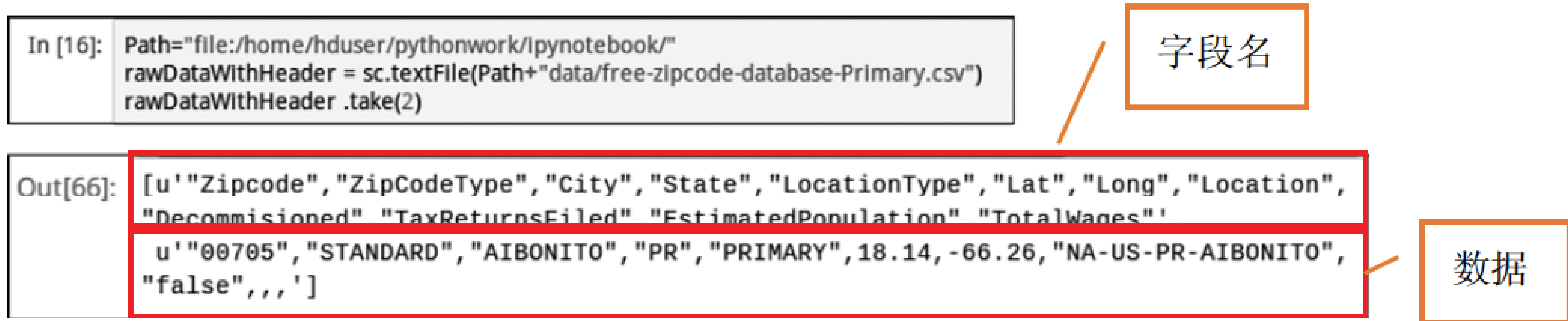


图 19-30 读取并查看数据

以上运行结果的第一项数据是字段名，所以我们必须在下一步骤去除表头。

步骤 04 删除第一项数据

因为第一项数据是字段名，所以我们要先删除：

```
In [70]: header = rawDataWithHeader.first()
         rawData = rawDataWithHeader.filter(lambda x:x !=header)
         rawData.first()

Out[70]: u'"00705","STANDARD","AIBONITO","PR","PRIMARY",18.14,-66.26,"NA-US-PR-A
         IBONITO","false",,,'
```

以上运行结果显示字段名已经被删除。程序代码详细说明如表 19-9 所示。

表 19-9

程序代码	说明
header = rawDataWithHeader.first()	以 .first() 取得第一项数据，存至 header 变量
rawData = rawDataWithHeader .filter(lambda x:x !=header)	以 .filter 配合 lambda 语句，创建匿名函数 x 为参数，以 x !=header 筛选删除 header
rawData.first()	设置按照性别 gender 进行分组统计

步骤05 删除“"”符号

上一步骤运行结果的每一个文字字段前后都有“"”符号，我们将在下一步骤删除双引号“"”。以下程序代码用 rawData.map( ) 处理每一项数据，并在 lambda 匿名函数中使用 x.replace("\"", "")删除“"”符号：

```
In [71]:  rDataI=rawData.map(lambda x: x.replace("\"", ""))
          rData.first()
Out[71]:  u'00705,STANDARD,AIBONITO,PR,PRIMARY,18.14,-66.26,NA-US-PR-AIBONITO,false,,,'
```

从以上运行结果可以看出双引号“"”已经删除。另外，还可以发现每一个字段都以“,”符号作为分隔，我们将在下一步骤获取每一个字段。

步骤06 获取每一个字段

接下来，我们用 rData.map 处理每一项数据，并在 lambda 匿名函数中使用 x.split(",")以“,”符号分隔来获取每一个字段：

```
In [72]:  ZipRDD = rData.map(lambda x: x.split(","))
          ZipRDD.first()
Out[72]:  [u'00705',
           u'STANDARD',
           u'AIBONITO',
           u'PR',
           u'PRIMARY',
           u'18.14',
           u'-66.26',
           u'NA-US-PR-AIBONITO',
           u'false',
           u'',
           u'',
           u'']
```

19.10.2 创建 zipcode_tab

我们已经创建 ZipRDD，接下来使用此 RDD 创建 DataFrames。

步骤01 创建 ZipCode Row 的 Schema

在创建 DataFrames 之前，我们必须使用 Row 创建每一个字段名与数据类型。

```
In [19]:  from pyspark.sql import Row
          zipcode_data =ZipRDD .map(lambda p:
            Row(
             zipcode=int(p[0]),
             zipCodeType=p[1],
             city=p[2],
             state=p[3]
             )
          )
          zipcode_data.take(5)
```

```
Out[19]: [Row(city=u'AIBONITO', state=u'PR', zipCodeType=u'STANDARD', zipcode=705),
          Row(city=u'ANASCO', state=u'PR', zipCodeType=u'STANDARD', zipcode=610),
          Row(city=u'ANGELES', state=u'PR', zipCodeType=u'PO BOX', zipcode=611),
          Row(city=u'ARECIBO', state=u'PR', zipCodeType=u'STANDARD', zipcode=612),
          Row(city=u'ADJUNTAS', state=u'PR', zipCodeType=u'STANDARD', zipcode=601)]
```

以上程序代码的详细说明如表 19-10 所示。

表 19-10　程序代码说明

程序代码	说明
from pyspark.sql import Row	导入 Row 模块
zipcode_data = ZipRDD.map(lambda p:	使用 ZipRDD.map 处理每一项数据，以 lambda 语句创建匿名函数传入 p 参数（每一项数据）
Row(zipcode=int(p[0]), zipCodeType=p[1], city=p[2], state=p[3]))	在匿名函数中，使用 Row 传入每一个字段，创建 Row 数据类型。其中，p[0] 是数字，转换为 int；其余是 String，不用转换
zipcode_data.take(5)	查看前 5 项 zipcode

步骤 02 创建 ZipCode Row 的 Schema

我们已经将数据转换为 Row 数据类型，之后就可以用 Row 数据类型创建 DataFrames 了：

```
In [72]: zipcode_df = sqlContext.createDataFrame(zipcode_data )
         zipcode_df.printSchema()

         root
          |-- city: string (nullable = true)
          |-- state: string (nullable = true)
          |-- zipCodeType: string (nullable = true)
          |-- zipcode: long (nullable = true)
```

以上程序代码说明如下：

- 首先，使用 sqlContext. createDataFrame( ) 方法传入 zipcode_data 数据，创建 DataFrame。
- 然后，使用 .printSchema() 查看 DataFrames 的 Schema。

步骤 03 创建登录临时表

之前已经创建了 DataFrames。接下来，我们将使用 DataFrames 登录临时表，然后查看前 10 项数据：

```
In [75]: zipcode_df.registerTempTable("zipcode_table")
         zipcode_df.show(10)

+---------+-----+-----------+-------+
|     city|state|zipCodeType|zipcode|
+---------+-----+-----------+-------+
| AIBONITO|   PR|   STANDARD|    705|
|   ANASCO|   PR|   STANDARD|    610|
|  ANGELES|   PR|     PO BOX|    611|
|  ARECIBO|   PR|   STANDARD|    612|
| ADJUNTAS|   PR|   STANDARD|    601|
| CASTANER|   PR|     PO BOX|    631|
|   AGUADA|   PR|   STANDARD|    602|
|AGUADILLA|   PR|   STANDARD|    603|
|AGUADILLA|   PR|     PO BOX|    604|
|AGUADILLA|   PR|     PO BOX|    605|
+---------+-----+-----------+-------+
only showing top 10 rows
```

19.10.3 Spark SQL 联接 zipcode_table 数据表

前一小节，我们已经登录了 zipcode_df 临时表，还有之前登录的 user_table 暂存数据表。我们可以将这两个数据表进行联接。

步骤01 查看纽约的用户数据

以下程序代码用 user_table 左联接 zip_table，查看纽约州的用户数据：

```
In [77]: sqlContext.sql("""
         SELECT u.* ,z.city,z.state
         FROM user_table u
         LEFT JOIN zipcode_table z  ON u.zipcode = z.zipcode
         WHERE z.state='NY'
         """).show(10)

+---+------+-------------+------+-------+----------+-----+
|age|gender|   occupation|userid|zipcode|      city|state|
+---+------+-------------+------+-------+----------+-----+
| 45|     M|administrator|    48|  12550|  NEWBURGH|   NY|
| 52|     F|    librarian|   204|  10960|     NYACK|   NY|
| 42|     M|        other|   766|  10960|     NYACK|   NY|
| 35|     F|        other|   760|  14211|   BUFFALO|   NY|
| 32|     F|        other|   155|  11217|  BROOKLYN|   NY|
| 30|     F|       writer|   557|  11217|  BROOKLYN|   NY|
| 27|     M|    marketing|   806|  11217|  BROOKLYN|   NY|
| 35|     F|     educator|   450|  11758|MASSAPEQUA|   NY|
| 28|     F|      student|   230|  14476|   KENDALL|   NY|
| 23|     M|administrator|   509|  10011|  NEW YORK|   NY|
+---+------+-------------+------+-------+----------+-----+
only showing top 10 rows
```

以上 SQL 语句的详细说明如表 19-11 所示。

表 19-11 SQL 语句说明

SQL 语法	说明
SELECT u.,z.city,z.state	显示 user_table 所有字段，再加上 zipcode_table 的 city 与 state 字段
FROM user_table u	读取 user_table 并取别名为 u
LEFT JOIN zipcode_table z	LEFT JOIN 联接 zipcode_table 并取别名为 z
ON u.zipcode = z.zipcode	联接条件 u.zipcode = z.zipcode
WHERE z.state='NY'	查询条件 state 是纽约

步骤 02　按照州统计

有了联接数据表，除了可以查询某一个州的用户，还可以按照州来统计。

```
In [93]: sqlContext.sql("""
         SELECT z.state ,count(*)
         FROM user_table u
         LEFT JOIN zipcode_table z  ON u.zipcode = z.zipcode
         GROUP BY z.state
         """).show(60)

         +-----+--------+
         |state|count(1)|
         +-----+--------+
         |   AZ|      14|
         |   SC|      11|
         |   LA|       6|
         |   MN|      78|
         |   NJ|      18|
         |   DC|      14|
         |   OR|      20|
```

以上 SQL 语句的详细说明如表 19-12 所示。

表 19-12　SQL 语句说明

SQL 语法	说明
SELECT z.state ,count(*)	显示 state 州名与数量总和
FROM user_table u	读取 user_table 并取别名为 u
LEFT JOIN zipcode_table z	LEFT JOIN 联接 zipcode_table 并取别名为 z
ON u.zipcode = z.zipcode	设置联接条件 u.zipcode = z.zipcode
GROUP BY z.state	按照 z.state 进行分组统计

19.10.4　DataFrame user_df 联接 zipcode_df

之前我们使用 Spark SQL 的临时表进行联接，事实上我们也可以使用 DataFrame 进行联接。以下范例，我们要将两个 DataFrames 进行联接。如图 19-31 所示，user_df 联接 zipcode_df，联接的结果是创建另外一个 DataFrames，即 joined_df。

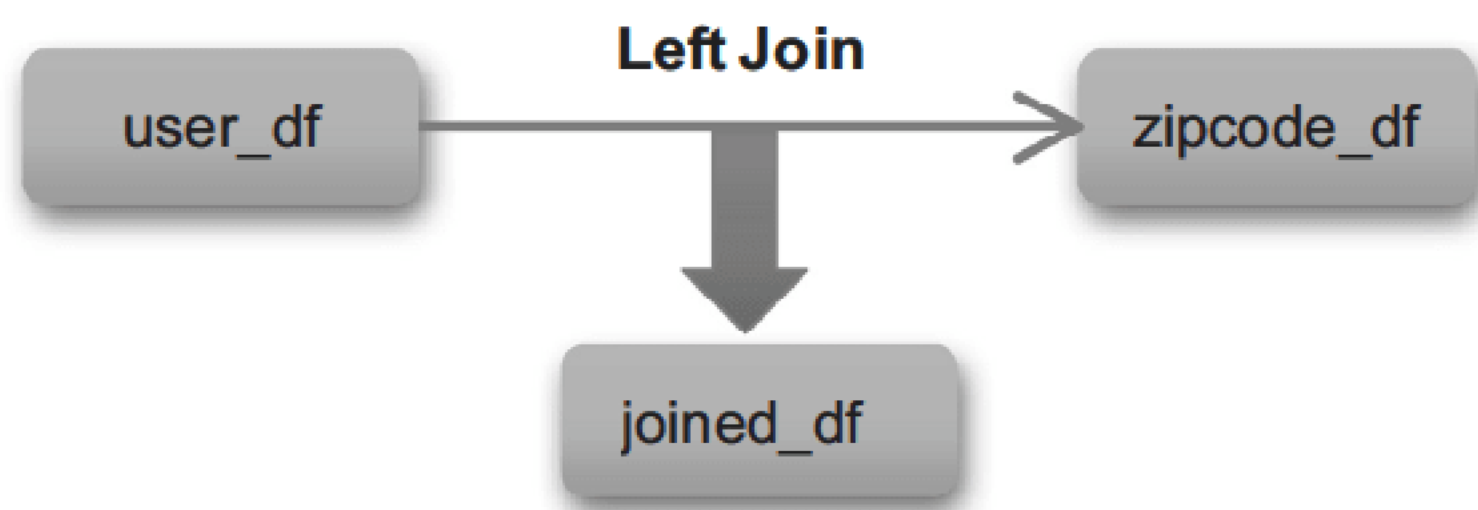


图 19-31　对 user_df、zipcode_df 进行联接示意图

步骤 01　user_df 联接 zipcode_df

接下来，将 user_df 联接 zipcode_df，联接的结果会创建另外一个 DataFrames（joined_

df），如图 19-32 所示。

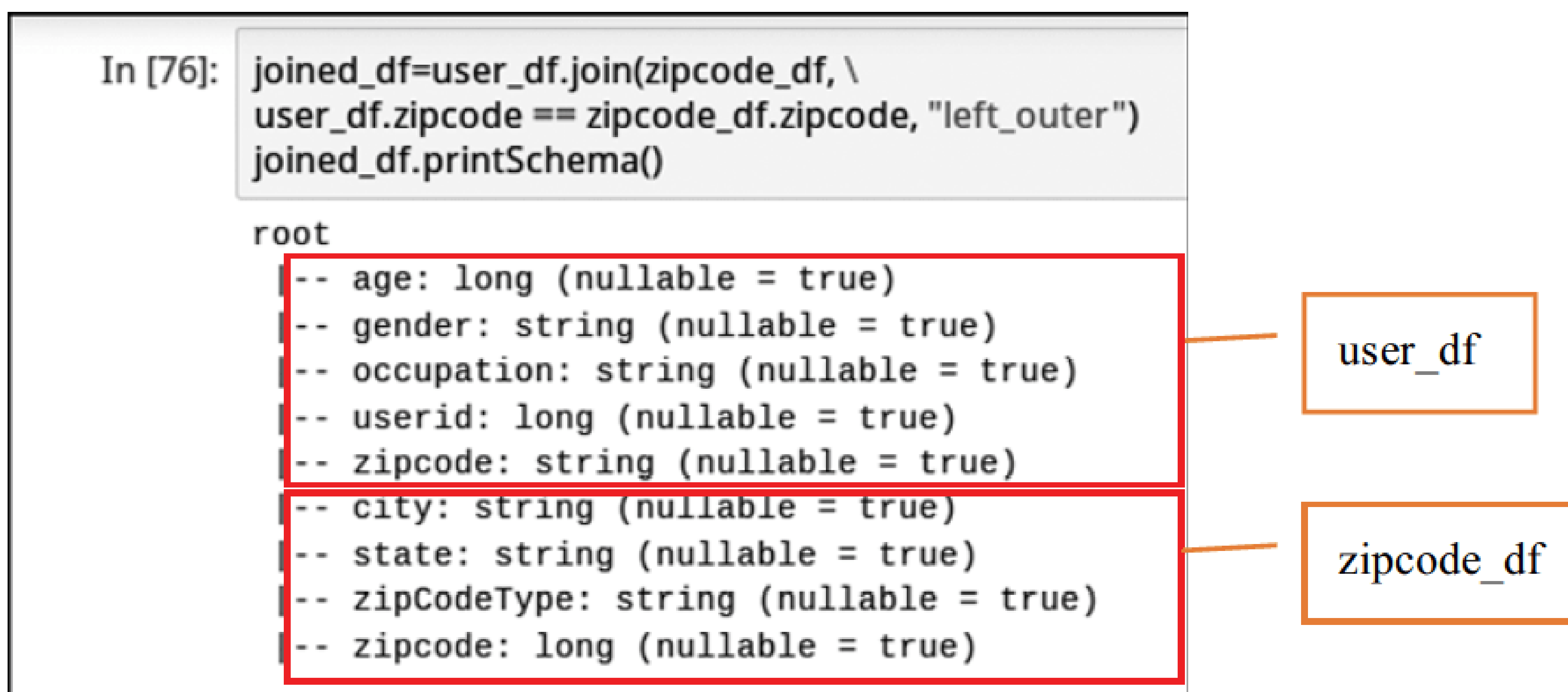


图 19-32 user_df 联接 zipcode_df

从图 19-32 可以看出 joined_df 是由 user_df 与 zipcode_df 组成的。以上程序代码说明如表 19-13 所示。

表 19-13 程序代码说明

程序代码	说明
joined_df=user_df.join(zipcode_df,	user_df 联接 zipcode_df 创建 joined_df
user_df.zipcode == zipcode_df.zipcode	设置联接条件
"left_outer"	设置联接的方式是 "left_outer"
joined_df.printSchema()	打印 joined_df 的 schema

步骤 02 查看 joined_df dataframe

查看 joined_df dataframe 显示前 10 项数据：

```
In [81]: joined_df.show(10)

+---+------+-------------+------+-------+-------------+-----+-----------+-------+
|age|gender|   occupation|userid|zipcode|         city|state|zipCodeType|zipcode|
+---+------+-------------+------+-------+-------------+-----+-----------+-------+
| 53|     F|        other|     2|  94043|MOUNTAIN VIEW|   CA|   STANDARD|  94043|
| 53|     M|       lawyer|    10|  90703|     CERRITOS|   CA|   STANDARD|  90703|
| 23|     M|       writer|     3|  32067|  ORANGE PARK|   FL|     PO BOX|  32067|
| 42|     M|    executive|     6|  98101|      SEATTLE|   WA|   STANDARD|  98101|
| 36|     M|administrator|     8|  05201|   BENNINGTON|   VT|   STANDARD|   5201|
| 24|     M|   technician|     4|  43537|       MAUMEE|   OH|   STANDARD|  43537|
| 39|     F|        other|    11|  30329|      ATLANTA|   GA|   STANDARD|  30329|
| 24|     M|   technician|     1|  85711|       TUCSON|   AZ|   STANDARD|  85711|
| 57|     M|administrator|     7|  91344|GRANADA HILLS|   CA|   STANDARD|  91344|
| 33|     F|        other|     5|  15213|   PITTSBURGH|   PA|   STANDARD|  15213|
+---+------+-------------+------+-------+-------------+-----+-----------+-------+
only showing top 10 rows
```

步骤 03 筛选纽约用户数据

我们可以将 .filter 用于筛选纽约用户数据：



```
In [82]: joined_df.filter("state='NY' ").show(10)

+---+------+-------------+------+-------+----------+-----+-----------+-------+
|age|gender|   occupation|userid|zipcode|      city|state|zipCodeType|zipcode|
+---+------+-------------+------+-------+----------+-----+-----------+-------+
| 45|     M|administrator|    48|  12550|  NEWBURGH|   NY|   STANDARD|  12550|
| 52|     F|    librarian|   204|  10960|     NYACK|   NY|   STANDARD|  10960|
| 42|     M|        other|   766|  10960|     NYACK|   NY|   STANDARD|  10960|
| 35|     F|        other|   760|  14211|   BUFFALO|   NY|   STANDARD|  14211|
| 32|     F|        other|   155|  11217|  BROOKLYN|   NY|   STANDARD|  11217|
| 30|     F|       writer|   557|  11217|  BROOKLYN|   NY|   STANDARD|  11217|
| 27|     M|    marketing|   806|  11217|  BROOKLYN|   NY|   STANDARD|  11217|
| 35|     F|     educator|   450|  11758|MASSAPEQUA|   NY|   STANDARD|  11758|
| 28|     F|      student|   230|  14476|   KENDALL|   NY|   STANDARD|  14476|
| 23|     M|administrator|   509|  10011|  NEW YORK|   NY|   STANDARD|  10011|
+---+------+-------------+------+-------+----------+-----+-----------+-------+
only showing top 10 rows
```

步骤 04 按照州分组统计

我们可以使用 .groupBy("state") 执行州分组统计：

```
In [94]: GroupByState_df=joined_df.groupBy("state").count()
         GroupByState_df.show(60)

+-----+-----+
|state|count|
+-----+-----+
|   AZ|   14|
|   SC|   11|
|   LA|    6|
|   MN|   78|
|   NJ|   18|
|   DC|   14|
|   OR|   20|
```

19.11 使用 Pandas DataFrames 绘图

Python 常用的数据处理 Pandas 套件也提供 DataFrames 功能。我们介绍如何将 Spark DataFrames 转换为 Pandas DataFrames，并使用 Pandas DataFrames 配合 matplotlib 模块，将数据绘制成直方图、圆饼图。

19.11.1 按照不同的州统计并以直方图显示

我们将之前创建的 GroupByState_df 转换为 Pandas DataFrames。

步骤 01 转换为 Pandas DataFrames

我们可以使用 .toPandas() 将 GroupByState_df 转换为 Pandas DataFrames，并设置索引是 state 州字段：

```
In [80]: Import pandas as pd
         GroupByState_pandas_df =GroupByState_df.toPandas().set_index('state')
         GroupByState_pandas_df
```

Out[80]:

	count
state	
MS	3
MT	2
TN	12
AE	1
NC	19
ND	2

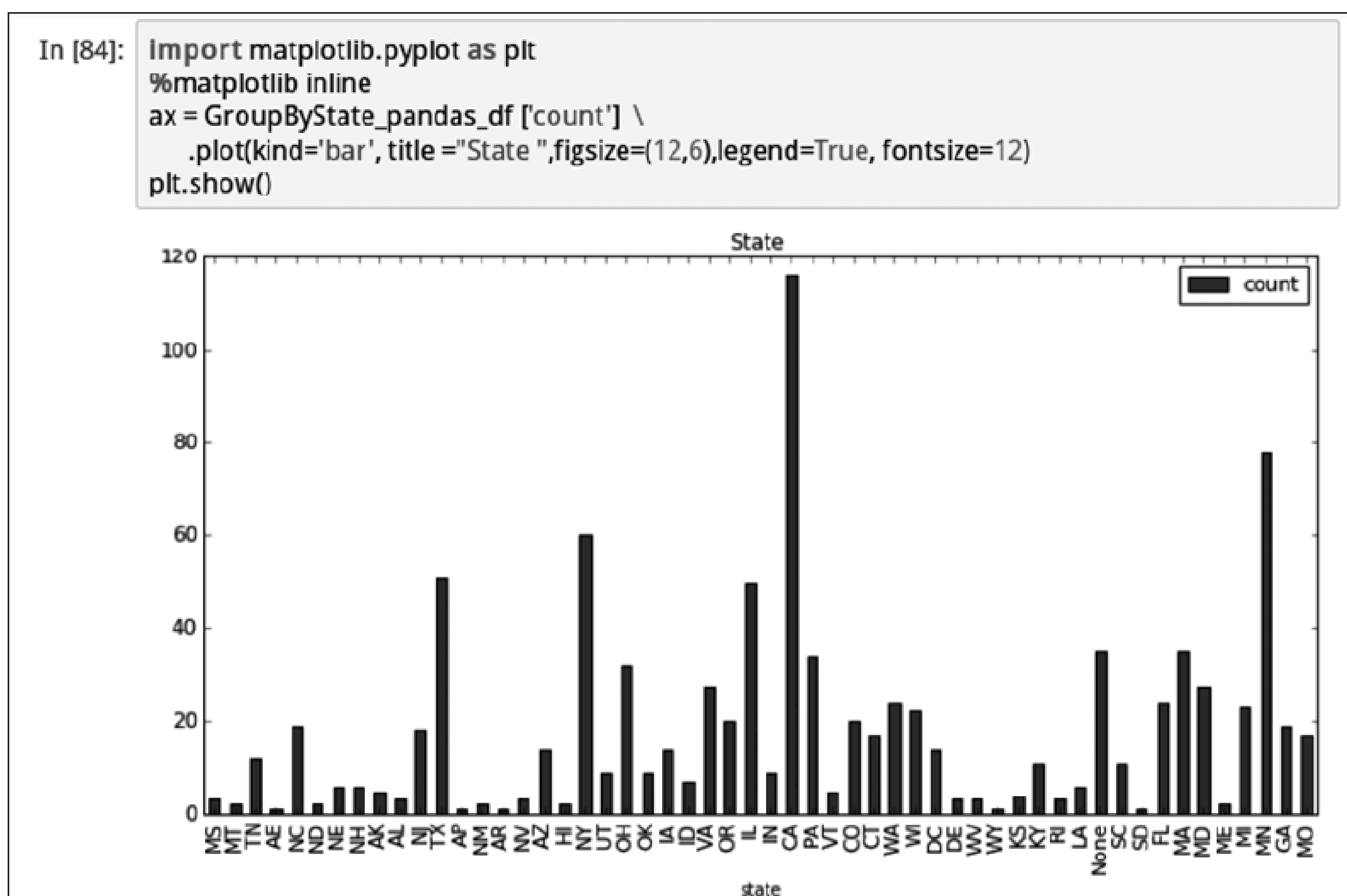
以上运行结果中有 54 项数据，不过这里只显示前 6 项。以上程序代码详细说明如表 19-14 所示。

表 19-14　SQL 语法说明

SQL 语法	说明
import pandas as pd GroupByState_pandas_df =	导入 pandas 模块
GroupByState_df.toPandas()	GroupByState_df 使用 .toPandas() 转换为 Pandas DataFrames
.set_index('state')	使用 set_index 设置 Pandas DataFrames 的索引为 state
GroupByState_pandas_df	查看 Pandas DataFrame GroupByState_pandas_df

步骤02 用 Pandas DataFrames 绘出直方图 Bar Chart

接下来，我们使用之前创建的按照州统计用户人数的 Pandas Dataframes，配合 matplotlib 模块将数据绘成直方图：



```
In [84]: Import matplotlib.pyplot as plt
         %matplotlib inline
         ax = GroupByState_pandas_df ['count'] \
             .plot(kind='bar', title ="State ",figsize=(12,6),legend=True, fontsize=12)
         plt.show()
```


以上程序代码的详细说明如表 19-15 所示。

表 19-15　程序代码说明

程序代码	说明
import matplotlib.pyplot as plt	导入 matplotlib.pyplot 模块
%matplotlib inline	将图形显示在 Ipython Notebook 中。如果不加入此命令，系统会将图形绘制在其他窗口中
ax = GroupByState_pandas_df ['counts'] .plot(kind='bar', title ="State", figsize=(12,6), legend=True, fontsize=12)	设置要绘图的是 counts 计算总和字段 绘图种类是 bar chart 直方图 图形的标题是 State 设置图形大小 设置要显示图例 设置字号
plt.show()	开始绘图

19.11.2　按照不同的职业统计人数并以圆饼图显示

接下来，我们将按照不同的职业统计人数并以圆饼图显示。

步骤 01 创建 Occupation_df

我们使用 sqlContext.sql() 输入 SQL 语句，按照不同的职业统计人数并创建 Occupation_df DataFrame。（因为项数很多，所以以下运行结果只显示部分界面。）

```
In [87]: Occupation_df=sqlContext.sql("""
SELECT u.occupation ,count(*) counts
FROM user_table u
GROUP BY occupation
""")
Occupation_df.show(30)

+-------------+------+
|   occupation|counts|
+-------------+------+
|    librarian|    51|
|      retired|    14|
|       lawyer|    12|
|         none|     9|
|       writer|    45|
|   programmer|    66|
```

步骤 02 创建 Occupation_pandas_df

我们使用 Occupation_df.toPandas() 转换为 Pandas DataFrames 并查看 Occupation_pandas_df。（因为项数很多，以下运行结果只显示部分界面。）

```
In [86]: Occupation_pandas_df =Occupation_df.toPandas().set_index('occupation')
         Occupation_pandas_df
```

Out[86]:

	counts
occupation	
administrator	79
programmer	66
salesman	12
doctor	7
scientist	31

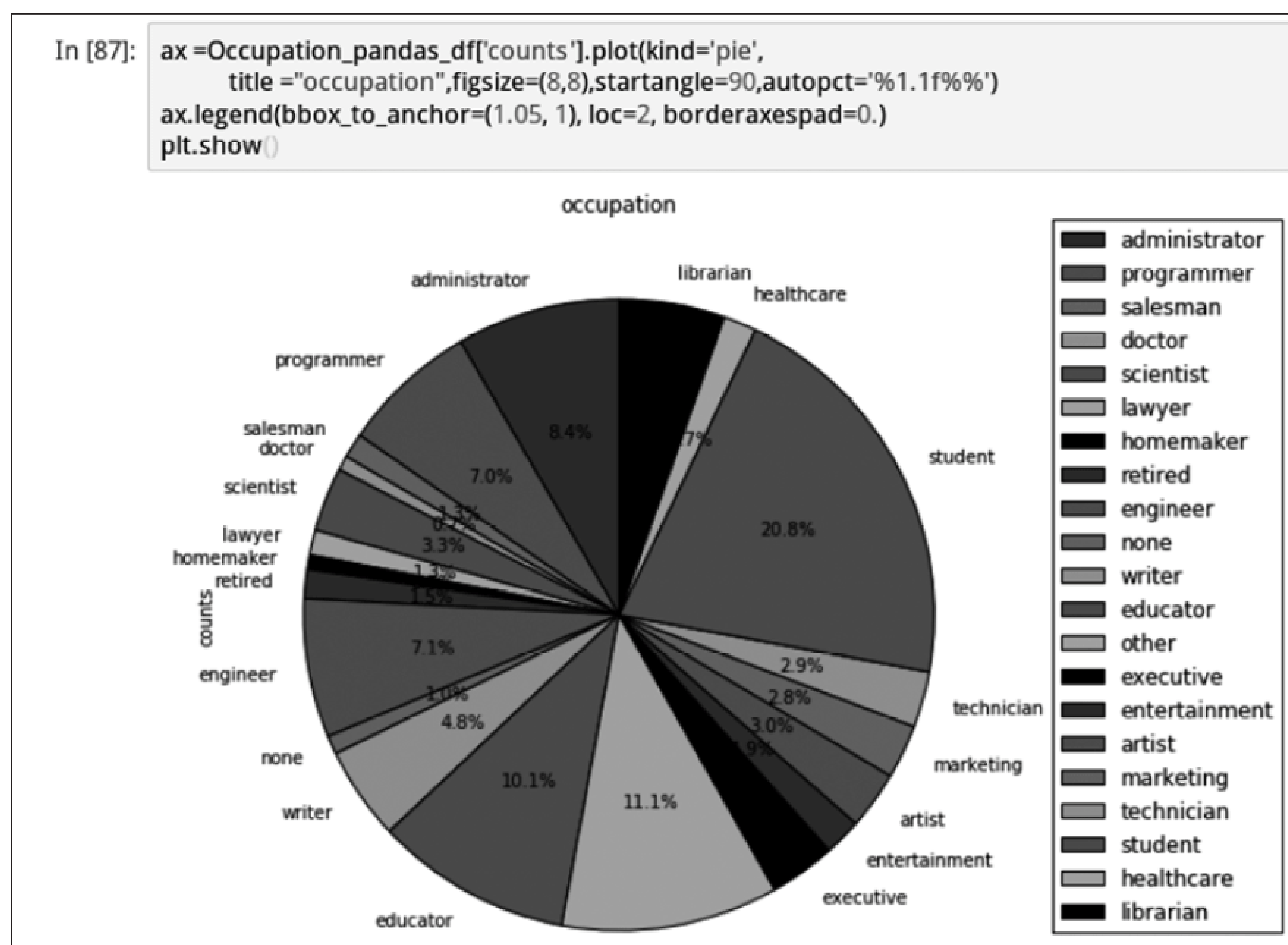
以上程序代码的详细说明如表 19-16 所示。

表 19-16　SQL 语句说明

SQL 语法	说明
Occupation_pandas_df = Occupation_df.toPandas() .set_index('occupation')	Occupation_df 使用 .toPandas() 转换为 Pandas DataFrames Occupation_pandas_df 使用 set_index 设置 Pandas DataFrames 的索引是 occupation 职业字段
Occupation_pandas_df	查看 Pandas DataFrames Occupation_pandas_df

步骤 03　用 Pandas DataFrames 绘出圆饼图 PieChart

按照不同的职业统计用户人数并以圆饼图显示：



```
In [87]: ax =Occupation_pandas_df['counts'].plot(kind='pie',
              title ="occupation",figsize=(8,8),startangle=90,autopct='%1.1f%%')
         ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.show()
```


以上程序代码的详细说明如表 19-17 所示。

表 19-17 程序代码说明

程序代码	说明
ax =Occupation_pandas_df['counts']. plot(kind='pie', title="occupation", figsize=(8,8), startangle=90, autopct='%1.1f%%')	设置要绘图的是 counts 计算总和字段 绘图种类是圆饼图 PieChart 图形的标题是 occupation 设置图形大小 设置图形旋转角度 设置显示圆饼图 %。
ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)	图例显示在右边
plt.show()	开始绘图

19.12 结论

本章介绍了 Python Spark RDD、DataFrames 与 Spark SQL 基本命令，并使用 DataFrames 与 Spark SQL 进行数据统计。在第 20~22 章，我们将介绍 Spark ML pipeline: Dataframes- based 机器学习 API。

第 20 章 Spark ML Pipeline 机器学习流程二元分类

本章将使用第 13 章二元分类分析 StumbleUpon 数据集示范如何使用 Spark ML Pipeline 机器学习流程二元分类，以决策树分类预测网页是暂时性的（ephemeral）还是长青的（evergreen），并且使用训练验证与交叉验证找出最佳模型，提高预测准确度。最后还将介绍如何使用随机森林分类（RandomForestClassifier）算法进一步提高准确率。

➢ Spark ML Pipeline 机器学习流程

Spark 机器学习工作流程（ML Pipeline ）的原理就是将机器学习的每一个阶段（例如数据处理、进行训练与测试、建立 Pipeline 流程）形成机器学习工作流程。流程示意图如图 20-1 所示。

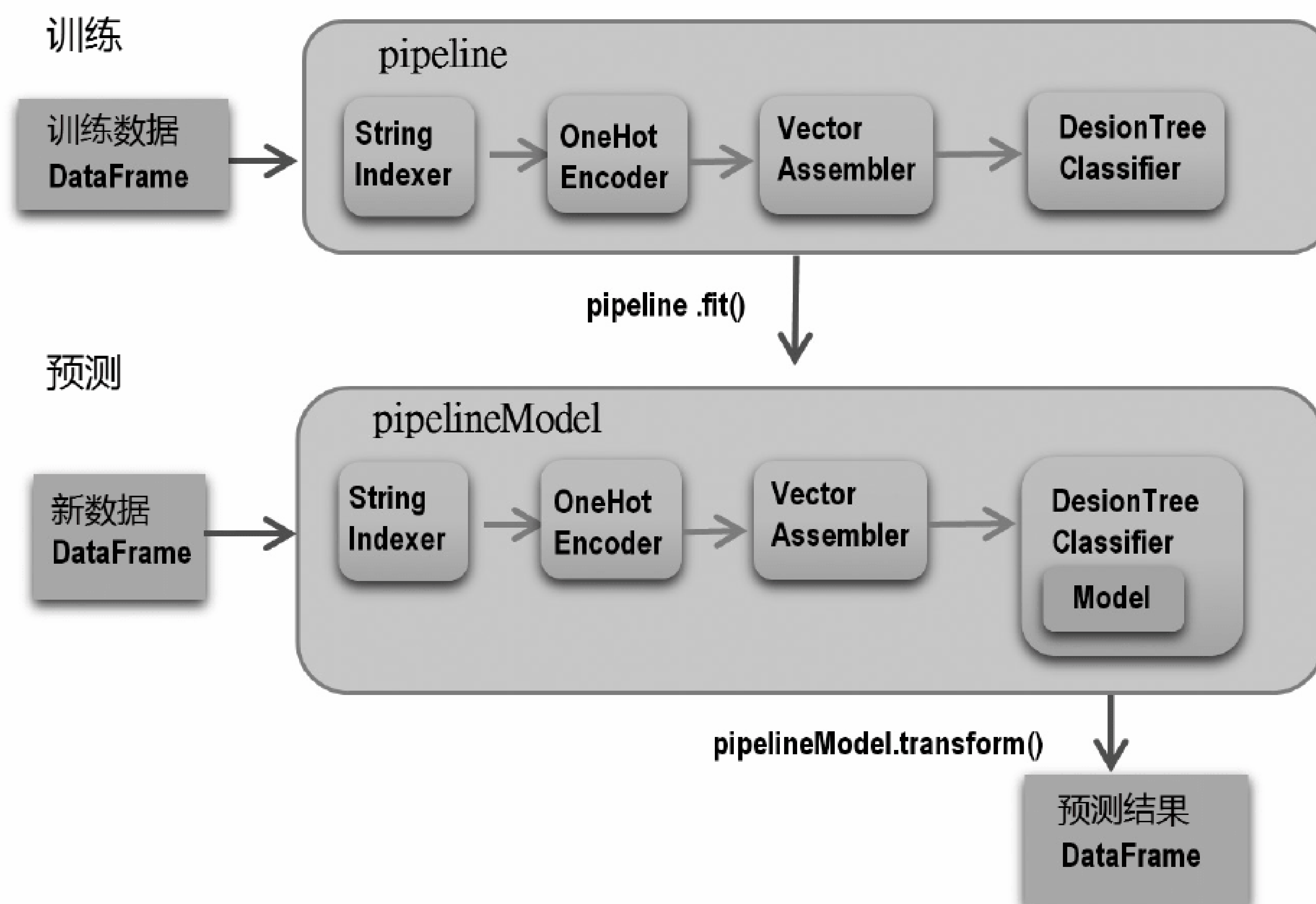


图 20-1　Spark ML Pipeline 机器学习流程示意图

对图 20-1 的说明如下：

（1）**建立机器学习流程 pipeline：**包含 4 个阶段（stages），前 3 个阶段是数据处理，第 4 阶段是 DesionTreeClassifier 机器学习分类算法。

- StringIndexer：将文字的分类特征字段转换为数字。
- OneHotEncoder：将一个数字的分类特征字段转为多个字段。
- VectorAssembler：将所有的特征字段整合成一个 Vector 字段。
- DesionTreeClassifier：进行训练并且产生模型。

（2）**训练：**“训练数据 DataFrame”使用 pipeline.fit() 进行训练。系统会按照顺序执行每一个阶段，最后产生 pipelineModel 模型。pipelineModel 与 pipeline 类似，只是多了训练后建立的模型 Model。

（3）预测：“新数据 DataFrame”使用 pipelineModel.transform() 进行预测。系统会按照顺序执行每一个阶段，并使用 DecisionTree Classifier Model 进行预测。预测完成后，会产生“预测结果 DataFrame”。

➢ **Spark 机器学习流程（ML Pipeline）的名词说明如下：**

- **DataFrame:** Spark ML 机器学习 API 处理的数据格式是 DataFrame，我们必须使用 Dataframe 存储训练数据、测试数据，最后预测结果也是 DataFrame。我们可以使用 sqlContext 读取文本文件创建 DataFrame，或将 RDD 转换为 DataFrame，也可以使用 Spark SQL 来操作。DataFrame 可以存储不同的数据类型，文字、特征字段所创建的向量 vectors、label 标签字段。
- **Transformer:** 是一个算法，可以使用 transform 方法将一个 DataFrame 转换为另一个 DataFrame。
- **Estimator:** 是一个算法，可以使用 fit 方法传入 DataFrame，产生一个 Transformer。
- **Pipeline**：可以串联多个 Transformers 与 Estimators 建立 ML 机器学习的 workflow 工作流程。
- **Parameter**：以上 Transformers 与 Estimators 都可以共享相同的 Parameter API。

20.1 数据准备

我们将使用 IPython Notebook 示范，因为使用 IPython Notebook 具有互动性的好处，可以看到指令执行后的结果。以下示范在本地运行，读者也可以参考第 9.9 节的说明，在不同的模式运行 IPython Notebook。读者可以参考本书附录 A 中有关本书范例程序下载与安装的说明，下载本章 IPython Notebook 的范例文件来进行练习。

20.1.1 在 local 模式执行 IPython Notebook

在“终端”程序中输入下列命令：

➢ **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

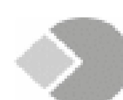
➢ **在本地运行 pyspark 使用 IPython Notebook 界面**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888，进去后就可以看到 IPython NoteBook 的界面。

步骤 01 配置文件读取路径

在 IPython Notebook 使用下列命令配置文件读取路径：



```
In [1]: global Path
        if sc.master[0:5]=="local" :
          Path="file:/home/hduser/pythonwork/PythonProject/"
        else:
          Path="hdfs://master:9000/user/hduser/"
```

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，代表当前是本地运行，读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，也就是在 cluster 运行，就有可能是 YARN client 或 Spark stand alone，读取 HDFS 文件。

步骤02 读取文本文件并查看数据项数

下列指令使用 sqlContext.read 导入文本文件，并创建 row_df DataFrame。

```
In [13]: row_df = sqlContext.read.format("csv") \
           .option("header", "true") \
           .option("delimiter", "\t") \
           .load(Path+"data/train.tsv")
         print row_df.count()

         7395
```

- .format("csv") 指定为 csv 格式。
- .option("header", "true") 指定第一行是字段名。
- .option("delimiter", "\t") 指定字段的分隔符是 tab 键 "\t"。
- .load(Path+"data/train.tsv") 指定要加载文件。
- 导入完成后，以 print row_df.count()打印项数。

以上运行结果显示共计 7395 项数据。我们可以看到 sqlContext.read 导入文本文件创建 DataFrame，比第 13.4.2 小节用 sc.textfile 导入文本文件创建 RDD 简单得多。

步骤03 查看Schema

可以使用 printSchema() 查看导入数据的 Schema。我们可以看到所有字段都是 string 数据格式，因为字段较多，这里只截取部分界面，如图 20-2 所示。

```
In [29]: row_df.printSchema()

root
 |-- url: string (nullable = true)
 |-- urlid: string (nullable = true)
 |-- boilerplate: string (nullable = true)
 |-- alchemy_category: string (nullable = true)
 |-- alchemy_category_score: string (nullable = true)
 |-- avglinksize: string (nullable = true)
 |-- commonlinkratio_1: string (nullable = true)
 |-- commonlinkratio_2: string (nullable = true)
 |-- commonlinkratio_3: string (nullable = true)
 |-- commonlinkratio_4: string (nullable = true)
```

所有字段都是 string 数据格式

图 20-2 查看Schema

步骤04 查看前10 项数据

DataFrames 字段数据太多，我们使用 Select 选取要显示的字段，然后查看前 10 项数据，如图 20-3 所示。

```
In [38]: row_df.select('url','alchemy_category','alchemy_category_score','is_news','label').show(10)

+--------------------+------------------+----------------------+-------+-----+
|                 url|  alchemy_category|alchemy_category_score|is_news|label|
+--------------------+------------------+----------------------+-------+-----+
|http://www.bloomb...|          business|              0.789131|      1|    0|
|http://www.popsci...|        recreation|              0.574147|      1|    1|
|http://www.menshe...|            health|              0.996526|      1|    1|
|http://www.dumbli...|            health|              0.801248|      1|    1|
|http://bleacherre...|            sports|              0.719157|      1|    0|
|http://www.conven...|                 ?|                     ?|      ?|    0|
|http://gofashionl...|arts_entertainment|               0.22111|      1|    1|
|http://www.inside...|                 ?|                     ?|      ?|    0|
|http://www.valetm...|                 ?|                     ?|      1|    1|
|http://www.howswe...|                 ?|                     ?|      ?|    1|
+--------------------+------------------+----------------------+-------+-----+
only showing top 10 rows
```

很多数据是“?”

图 20-3　查看前 10 项数据

在图 20-3 中，我们可以看到很多都是“?”，下一小节将介绍如何处理。

20.1.2　编写 DataFrames UDF 用户自定义函数

DataFrames UDF 是 User Define Function 的简写，也就是用户自定义函数。我们将编写 UDF 将“?”转换为“0”。

步骤01 编写 DataFrames UDF 用户自定义函数

我们使用下列指令编写 DataFrames UDF 用户自定义函数。

```
In [41]: from pyspark.sql.functions import udf
         def replace_question(x):
           return ("0" if x=="?" else x)
         replace_question= udf(replace_question)
```

以上指令说明如下：

（1）首先从 pyspark.sql.functions 导入 udf 模块。

（2）然后用 def replace_question(x) 定义 python 函数。这个函数很简单，传入 x 参数，判断如果是问号“?”就转为“0”。

（3）最后使用 udf 方法将 replace_question 转换为 DataFrames UDF 用户自定义函数。

步骤02 导入相关模块

我们需要导入下列模块：



（1）从 pyspark.sql.functions 导入 col 模块，后续我们可以用此模块读取字段数据。
（2）另外，还需要导入 pyspark.sql.types 模块，作为后续转换数据类型使用。

```
In [1]:  from pyspark.sql.functions import col
         import pyspark.sql.types
```

步骤 03 使用 replace_question UDF 用户自定义函数

以下指令把 row_df DataFrames 第 4 个字段至最后一个字段转换为 double。其中，最后一个字段是 label，其余是 feature。

```
In [18]:  df= row_df.select(
                ['url','alchemy_category' ]+
                [replace_question(col(column)).cast("double").alias(column)
                 for column in row_df.columns[4:] ] )
```

以上指令说明如下：
（1）用 row_df.select 选取字段。
（2）选取字段 ['url','alchemy_category']，不需要转换。
（3）for column in row_df.columns[4:] 读取第 4 个字段至最后一个字段。
（4）col(column) 读取字段数据并且调用 replace_question 自定义函数删除问号“?”。
（5）.cast("double") 转换为 double。
（6）.alias(column) 把别名设置为原来的字段名。

步骤 04 查看使用 replace_question UDF 转换后的字段

转换后，前两个字段是 url 和 alchemy_category，其余字段都已经转换为 double。（因为字段太多，这里只显示部分界面，如图 20-4 所示。）

```
In [45]:  print df.printSchema()

root
 |-- url: string (nullable = true)
 |-- alchemy_category: string (nullable = true)
 |-- alchemy_category_score: double (nullable = true)
 |-- avglinksize: double (nullable = true)
 |-- commonlinkratio_1: double (nullable = true)
 |-- commonlinkratio_2: double (nullable = true)
 |-- commonlinkratio_3: double (nullable = true)
 |-- commonlinkratio_4: double (nullable = true)
 |-- compression_ratio: double (nullable = true)
 |-- embed_ratio: double (nullable = true)
```

已经转换为double

图 20-4　查看使用 replace_question UDF 转换后的字段

步骤 05 查看结果

以下使用 df.select 查看结果，我们会发现之前字段问号都已经转换为 0，如图 20-5 所示。

```
In [51]: df.select('url','alchemy_category','alchemy_category_score','is_news','label').show(10)

+--------------------+----------------+----------------------+-------+-----+
|                 url|alchemy_category|alchemy_category_score|is_news|label|
+--------------------+----------------+----------------------+-------+-----+
|http://www.bloomb...|        business|              0.789131|    1.0|  0.0|
|http://www.popsci...|      recreation|              0.574147|    1.0|  1.0|
|http://www.menshe...|          health|              0.996526|    1.0|  1.0|
|http://www.dumbli...|          health|              0.801248|    1.0|  1.0|
|http://bleacherre...|          sports|              0.719157|    1.0|  0.0|
|http://www.conven...|               ?|                   0.0|    0.0|  0.0|
|http://gofashionl...|arts_entertainment|             0.22111|    1.0|  1.0|
|http://www.inside...|               ?|                   0.0|    0.0|  0.0|
|http://www.valetm...|               ?|                   0.0|    1.0|  1.0|
|http://www.howswe...|               ?|                   0.0|    0.0|  1.0|
+--------------------+----------------+----------------------+-------+-----+
only showing top 10 rows
```

问号都已经转换为0

图 20-5　查看结果

20.1.3　将数据分成 train_df 与 test_df

使用 randomSplit 将数据按照 7 : 3 的比例分成 train_df（训练数据）与 test_df（测试数据），并且 .cache() 暂存在内存中，以加快后续程序运行的速度。

```
In [11]: train_df, test_df = df.randomSplit([0.7, 0.3])
         train_df.cache()
         test_df.cache()

Out[11]: DataFrame[url: string, alchemy_category: string, alchemy_category_score
         : double, avglinksize: double, commonlinkratio_1: double, commonlinkrat
         io_2: double, commonlinkratio_3: double, commonlinkratio_4: double, com
         pression_ratio: double, embed_ratio: double, framebased: double, frameT
         agRatio: double, hasDomainLink: double, html_ratio: double, image_ratio
         : double, is_news: double, lengthyLinkDomain: double, linkwordscore: do
         uble, news_front_page: double, non_markup_alphanum_characters: double,
         numberOfLinks: double, numwords_in_url: double, parametrizedLinkRatio:
         double, spelling_errors_ratio: double, label: double]
```

20.2　机器学习 pipeline 流程的组件

在建立 pipeline 数据处理流程之前，我们先介绍将用来建立流程的组件。

（1）StringIndexer：将文字的分类特征字段转换为数字。
（2）OneHotEncoder：将一个数值的分类特征字段转换为多个字段的 Vector。
（3）VectorAssembler：将多个特征字段整合成一个 Vector 的特征字段。
（4）DecisionTreeClassifier：决策树分类。

20.2.1　StringIndexer

StringIndexer 的功能类似于 categoriesMap，是网页分类特征字段的字典，可用于将字符串分类特征字段转换为数值。StringIndexer 是一个"Estimator"，所以使用上必须分为两个步骤：

（1）使用 fit 方法传入参数 DataFrame，产生一个“Transformer”。

（2）针对生成的“Transformer”，使用 transform 方法将 DataFrame 转换为另一个 DataFrame。

步骤 01 导入StringIndexer 模块

首先导入 pyspark.ml.feature. StringIndexer 模块。

```
In [97]: from pyspark.ml.feature import StringIndexer
```

步骤 02 创建 StringIndexer

以下程序创建“StringIndexer”，传入下列参数：

- inputCol：要转换的字段名。
- outputCol：转换后产生的字段名。

运行后创建“StringIndexer”的 categoryIndexer。

```
In [91]: from pyspark.ml.feature import StringIndexer
         categoryIndexer = StringIndexer(
                         inputCol='alchemy_category',
                         outputCol="alchemy_category_Index")
```

步骤 03 stringIndexer 使用 .fit 方法生成“Transformer”

之前步骤 StringIndexer 创建 categoryIndexer 是一个“Estimator”，所以使用 categoryIndexer. fit 方法传入参数 train_df 生成的一个“Transformer” categoryTransformer。

```
In [15]: categoryTransformer=categoryIndexer.fit(df)
```

步骤 04 查看 categoryTransformer 的内容

之前步骤生成的“Transformer” categoryTransformer 的 labels 属性其实就是一个网页分类的字典（或对照表），我们可以使用下列程序代码来查看 categoryTransformer.labels 的内容：

```
In [90]: for i in range(0,len(st_Transformer.labels)):
             print str(i)+':'+st_Transformer.labels[i]

         0:?
         1:recreation
         2:arts_entertainment
         3:business
         4:health
         5:sports
         6:culture_politics
         7:computer_internet
         8:science_technology
         9:gaming
         10:religion
         11:law_crime
         12:unknown
         13:weather
```

步骤05 使用 categoryTransformer 将所有 train_df 转换成 df1

接下来，我们可以用 categoryTransformer 将 train_df 转换成 df1。

```
In [94]: df1=categoryTransformer.transform(train_df)
```

步骤06 查看转换后的 df1 字段

使用 df1.columns 查看全部字段名，就可以发现新增了 'alchemy_category_Index' 字段，如图 20-6 所示。

```
In [95]: print df1.columns

['url', 'alchemy_category', 'alchemy_category_score', 'avglinksize', 'commo
nlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3', 'commonlinkratio_4
', 'compression_ratio', 'embed_ratio', 'framebased', 'frameTagRatio', 'hasD
omainLink', 'html_ratio', 'image_ratio', 'is_news', 'lengthyLinkDomain', 'l
inkwordscore', 'news_front_page', 'non_markup_alphanum_characters', 'number
OfLinks', 'numwords_in_url', 'parametrizedLinkRatio', 'spelling_errors_rati
o', 'label', 'alchemy_category_Index']
```

转换后新增的字段

图 20-6 查看转换后的 df1 字段

步骤07 查看转换后的结果

使用指令查看 df1 的网页分类字段转换前后的差异，如图 20-7 所示。

```
In [90]: df1.select("alchemy_category","alchemy_category_Index").show(10)

+-----------------+----------------------+
| alchemy_category|alchemy_category_Index|
+-----------------+----------------------+
|         business|                   3.0|
|       recreation|                   1.0|
|           health|                   4.0|
|           health|                   4.0|
|           sports|                   5.0|
|                ?|                   0.0|
|arts_entertainment|                  2.0|
|                ?|                   0.0|
|                ?|                   0.0|
|                ?|                   0.0|
+-----------------+----------------------+
only showing top 10 rows
```

" business " 转换为3

图 20-7 查看转换后的结果

我们可以看到转换前"alchemy_category"是 String 字符串，转换后的字段"alchemy_category_Index"是数值。例如，第一项 alchemy_category 的值"business" 转换为 3.0。

20.2.2 OneHotEncoder

OneHotEncoder 可以将一个数值的分类特征字段转换为多个字段的 Vector。

步骤01 导入OneHotEncoder 模块

首先从 pyspark.ml.feature 导入 OneHotEncoder 模块。



```
In [114]: from pyspark.ml.feature import OneHotEncoder
```

步骤 02 创建 OneHotEncoder

以下程序创建“OneHotEncoder”，传入下列参数：

- dropLast 设置为 False。
- inputCol 是要转换的字段名。
- outputCol 是转换后产生的字段名。

创建后的“OneHotEncoder”是 encoder 变量。

```
In [139]: encoder = OneHotEncoder(dropLast=False,
                      inputCol='alchemy_category_Index',
                      outputCol="alchemy_category_IndexVec")
```

步骤 03 OneHotEncoder 使用 transform 转换

OneHotEncoder 使用 transform 转换后结果是 df2，我们可以使用下列指令查看字段，如图 20-8 所示。

```
In [22]: df2=encoder.transform(df1)
         print df2.columns

['url', 'alchemy_category', 'alchemy_category_score', 'avg
linksize', 'commonlinkratio_1', 'commonlinkratio_2', 'comm
onlinkratio_3', 'commonlinkratio_4', 'compression_ratio',
'embed_ratio', 'framebased', 'frameTagRatio', 'hasDomainLi
nk', 'html_ratio', 'image_ratio', 'is_news', 'lengthyLinkD
omain', 'linkwordscore', 'news_front_page', 'non_markup_al
phanum_characters', 'numberOfLinks', 'numwords_in_url', 'p
arametrizedLinkRatio', 'spelling_errors_ratio', 'label', '
alchemy_category_Index', 'alchemy_category_IndexVec']
```

新增了 'alchemy_category_IndexVec' 字段

图 20-8　OneHotEncoder 使用 transform 转换

从以上运行结果可以看到新增了一个 'alchemy_category_IndexVec' 字段。

步骤 04 查看转换后新增的字段（见图 20-9）

```
In [141]: df2.select("alchemy_category","alchemy_...
                     "alchemy_category_IndexVec...

+-----------------+----------------------+-------------------------+
| alchemy_category|alchemy_category_Index|alchemy_category_IndexVec|
+-----------------+----------------------+-------------------------+
|         business|                   3.0|           (14,[3],[1.0])|
|       recreation|                   1.0|           (14,[1],[1.0])|
|           health|                   4.0|           (14,[4],[1.0])|
|           health|                   4.0|           (14,[4],[1.0])|
|           sports|                   5.0|           (14,[5],[1.0])|
|                ?|                   0.0|           (14,[0],[1.0])|
|arts_entertainment|                  2.0|           (14,[2],[1.0])|
|                ?|                   0.0|           (14,[0],[1.0])|
|                ?|                   0.0|           (14,[0],[1.0])|
|                ?|                   0.0|           (14,[0],[1.0])|
+-----------------+----------------------+-------------------------+
only showing top 10 rows
```

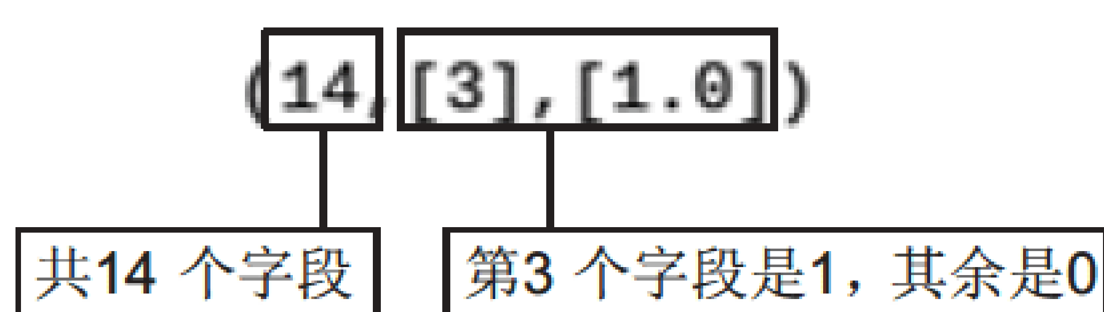
Business 先转为 3，再转为 vector(14, [3], [1.0])

图 20-9　查看转换后新增的字段

可以使用下列指令查看 3 个字段：

- alchemy_category：原本字符串的网页分类。
- alchemy_category_Index：转换为一个字段的数值。
- alchemy_category_IndexVec：转换为 14 个字段的数值。

转换后新增字段 alchemy_category_Index，例如第 1 项原本是 business，先转为 3，再转为 vector(14,[3], [1.0])。alchemy_category_IndexVec 的表示方式其实是 SparesVector 格式，其意义说明如下：



也就是说，第 3 个字段（从 0 开始算起第 3 个）是 1，其余是 0，即 0,0,0,1,0,0,0,0,0,0,0,0,0,0。使用 SparesVector 存储数据的好处是：因为大部分都是 0，所以只需要指定哪些字段不是 0，既可让数据更易读，也节省存储空间。

20.2.3 VectorAssembler

VectorAssembler 可以将多个特征字段整合成一个特征的 Vector。

步骤01 导入VectorAssembler 模块

首先导入 pyspark.ml.feature.VectorAssembler 模块。

```
In [24]: from pyspark.ml.feature import VectorAssembler
```

步骤02 创建全部特征字段 List

我们使用下列程序代码来创建全部特征字段名的 assemblerInputs：就是之前步骤生成的分类特征字段 alchemy_category_Index（14 个字段的数值），加上原本第 4 字段到倒数第 2 个字段的数值特征字段。我们可以用 print 查看 assemblerInputs，即所有的特征字段 List。

```
In [25]: assemblerInputs =['alchemy_category_IndexVec'] + row_df.columns[4:-1]
         print assemblerInputs

         ['alchemy_category_IndexVec', 'alchemy_category_score', 'avglin
         ksize', 'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkra
         tio_3', 'commonlinkratio_4', 'compression_ratio', 'embed_ratio'
         , 'framebased', 'frameTagRatio', 'hasDomainLink', 'html_ratio',
          'image_ratio', 'is_news', 'lengthyLinkDomain', 'linkwordscore'
         , 'news_front_page', 'non_markup_alphanum_characters', 'numberO
         fLinks', 'numwords_in_url', 'parametrizedLinkRatio', 'spelling_
         errors_ratio']
```

步骤03 创建 VectorAssembler

以下程序创建“VectorAssembler”，传入下列参数：



- inputCols 是要整合的所有字段名 assemblerInputs。
- outputCol 是转换后产生的字段名 features。

运行后创建“VectorAssembler”的 assembler。

```
In [26]: assembler = VectorAssembler(
                     inputCols=assemblerInputs,
                     outputCol="features")
```

步骤 04 运行 VectorAssembler 转换

有了 VectorAssembler，就可以运行 transform 转换产生 df3。

```
In [29]: df3=assembler.transform(df2)
```

步骤 05 查看整合后新增的字段

使用 df3.columns 查看全部字段名，我们可以发现新增了 features 字段，如图 20-10 所示。

```
In [28]: print df3.columns

['url', 'alchemy_category', 'alchemy_category_score', 'avglinks
ize', 'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkrati
o_3', 'commonlinkratio_4', 'compression_ratio', 'embed_ratio',
'framebased', 'frameTagRatio', 'hasDomainLink', 'html_ratio', '
image_ratio', 'is_news', 'lengthyLinkDomain', 'linkwordscore',
'news_front_page', 'non_markup_alphanum_characters', 'numberOfL
inks', 'numwords_in_url', 'parametrizedLinkRatio', 'spelling_er
rors_ratio', 'label', 'alchemy_category_Index', 'alchemy_catego
ry_IndexVec', 'features']
```

整合后新增的字段

图 20-10　查看整合后新增的字段

步骤 06 查看 features 特征字段

可以用下列指令查看 features 字段，这也是 SparesVector 格式，只是内容太多了，所以下面的截图无法看到全部内容。

```
In [29]: df3.select('features').show(5)

+--------------------+
|            features|
+--------------------+
|(36,[3,14,15,16,1...|
|(36,[1,14,15,16,1...|
|(36,[4,14,15,16,1...|
|(36,[4,14,15,16,1...|
|(36,[5,14,15,16,1...|
+--------------------+
only showing top 5 rows
```

步骤 07 查看 features 特征字段（见图 20-11）

为了能够看到 features 特征字段，我们使用 take(1)来查看（见图 20-11）。

```
In [30]: df3.select('features').take(1)
Out[30]: [Row(features=SparseVector(36, {3: 1.0, 14: 0.7891, 15: 2
.0556, 16: 0.6765, 17: 0.2059, 18: 0.0471, 19: 0.0235, 20
: 0.4438, 23: 0.0908, 25: 0.2458, 26: 0.0039, 27: 1.0, 28
: 1.0, 29: 24.0, 31: 5424.0, 32: 170.0, 33: 8.0, 34: 0.15
29, 35: 0.0791}))]
```

共36 个字段

图 20-11　查看 features 特征字段

从图 20-11 中，我们可以看到这是 SparesVector 数据格式，共 36 个字段，第 3 个字段是 1，第 14 个字段是 0.7891，第 15 个字段是 2.0556……，其余字段都是 0。

20.2.4　使用 DecisionTreeClassifier 二元分类

在之前的步骤中，我们已经准备好了数据：features 与 label 字段。接下来就可以使用 DecisionTreeClassifier 执行二元分类了。DecisionTreeClassifier 是一个“Estimator”，所以使用上必须分为以下两个步骤：

（1）使用 fit 方法，进行训练产生 DecisionTree 模型。

（2）DecisionTree 模型使用 .transform () 进行转换，转换后会产生预测结果。

步骤 01　导入模块

首先导入 pyspark.ml.classification .DecisionTreeClassifier 模块。

```
In [ ]: from pyspark.ml.classification import DecisionTreeClassifier
```

步骤 02　DecisionTreeClassifier

运行 DecisionTreeClassifier，传入下列参数，运行后存储在 dt 变量中：

- labelCol="label"，标签字段。
- featuresCol="features"，特征字段。
- impurity="gini", maxDepth=10, maxBins=14 决策树的参数，可以参考第 13 章说明。

```
In [133]: dt = DecisionTreeClassifier(labelCol="label", featuresCol="features",
                     impurity="gini",maxDepth=10, maxBins=14)
```

步骤 03　执行训练

之前创建的 dt 决策树分类，我们可以使用 .fit() 方法进行训练，训练结果产生 dt_model 模型，之后可以用 print 查看产生的模型。

```
In [37]: dt_model=dt.fit(df3)
         print dt_model

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4959b1
cc4d83f0347245) of depth 10 with 723 nodes
```

步骤04 进行预测

建立模型后就可以使用 .transform () 进行转换了，转换后会产生预测结果 df4。

```
In [71]: df4=dt_model.transform(df3)
```

关于决策树模型，后面章节还会详细介绍。

20.3 建立机器学习 pipeline 流程

之前已经介绍了机器学习 pipeline 流程的组件，介绍的过程步骤很多，只是为了方便解说而已，其实建立机器学习流程很简单。接下来我们会发现只需要数行程序语句就可以建立完成。

步骤01 导入模块

导入全部需要的模块。

```
In [52]: from pyspark.ml import Pipeline
         from pyspark.ml.feature import  StringIndexer, OneHotEncoder,VectorAssembler
         from pyspark.ml.classification import DecisionTreeClassifier
```

步骤02 建立 pipeline

建立 pipeline 的程序代码如下，机器学习流程组件 StringIndexer、OneHotEncoder、VectorAssembler、DesionTreeClassifier 之前都已经介绍了，只有最后一行程序代码使用 Pipeline() 传入 stages 阶段参数，将所有的组件变量名称按照顺序输入[stringIndexer, encoder, assembler, dt]。最后运行结果就建立了 pipeline，如图 20-12 所示。

```
In [94]: stringIndexer = StringIndexer(inputCol='alchemy_category',
                                       outputCol="alchemy_category_Index")
         encoder = OneHotEncoder(dropLast=False,
                                 inputCol='alchemy_category_Index',
                                 outputCol="alchemy_category_IndexVec")
         assemblerInputs =['alchemy_category_IndexVec'] + row_df.columns[4:-1]
         assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
         dt = DecisionTreeClassifier(labelCol="label", featuresCol="features",impurity="gini",
                                     maxDepth=10, maxBins=14)
         pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler,dt ])
```

建立pipeline

图 20-12　建立 pipeline

建立 pipeline 不需要花太多时间，但是后续 pipeline.fit 进行训练会花很多时间。

步骤03 查看pipeline 阶段

建立 pipeline 后，我们可以使用 getStages() 看到每一个阶段。

```
In [54]: pipeline.getStages()
Out[54]: [StringIndexer_4133a408061f8998cd81,
          OneHotEncoder_4b089b9019a9742aa6be,
          VectorAssembler_428089baaa017effb7aa,
          DecisionTreeClassifier_489f9ad8be0260ffad5f]
```

20.4 使用 pipeline 进行数据处理与训练

之前我们已经建立了机器学习流程 pipeline。下面可以使用 pipeline 进行数据处理与训练。

步骤01 使用 pipeline.fit 进行训练

以下程序代码使用 pipeline.fit 进行数据处理与训练，传入 train_df 训练数据，训练数据会执行 pipeline 的所有阶段 stringIndexer、encoder、assembler、dt，所以会比较花时间，最后产生的结果是 pipelineModel。

```
In [20]: pipelineModel = pipeline.fit(train_df)
```

步骤02 查看训练完成后的决策树模型

pipelineModel 的第 3 阶段会产生决策树的模型，我们可以用下列指令来查看这个模型。

```
In [97]: pipelineModel.stages[3]
Out[97]: DecisionTreeClassificationModel (uid=DecisionTreeClassifier_47ce8c
         c4972d5f67dd01) of depth 10 with 637 nodes
```

步骤03 查看训练完成后的决策树模型规则

我们还可以进一步使用 toDebugString 查看决策树模型的规则。

```
In [98]: print pipelineModel.stages[3].toDebugString
         DecisionTreeClassificationModel (uid=DecisionTreeClassifier_47ce8c
         c4972d5f67dd01) of depth 10 with 637 nodes
           If (feature 31 <= 1790.0)
            If (feature 4 in {1.0})
             If (feature 23 <= 0.06805293)
              If (feature 25 <= 0.238397021)
               If (feature 31 <= 1337.0)
                If (feature 25 <= 0.168502544)
                 Predict: 0.0
                Else (feature 25 > 0.168502544)
                 Predict: 1.0
```

20.5 使用 pipelineModel 进行预测

之前我们已经训练完成并建立模型，现在我们可以进行预测。

步骤 01 使用 pipelineModel.transform 进行预测

以下程序代码 pipelineModel 使用 transform 方法传入 test_df 测试数据，进行预测。

```
In [68]: predicted=pipelineModel.transform(test_df)
```

步骤 02 查看预测结果字段

查看预测后的 Schema，发现新增了 3 个字段，如图 20-13 所示。

```
In [90]: print predicted.columns

['url', 'alchemy_category', 'alchemy_category_score', 'avglinksize
', 'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3',
'commonlinkratio_4', 'compression_ratio', 'embed_ratio', 'framebas
ed', 'frameTagRatio', 'hasDomainLink', 'html_ratio', 'image_ratio'
, 'is_news', 'lengthyLinkDomain', 'linkwordscore', 'news_front_pag
e', 'non_markup_alphanum_characters', 'numberOfLinks', 'numwords_i
n_url', 'parametrizedLinkRatio', 'spelling_errors_ratio', 'label',
 'alchemy_category_Index', 'alchemy_category_IndexVec', 'features'
, 'rawPrediction', 'probability', 'prediction']
```

预测后新增的字段

图 20-13 查看预测结果字段

步骤 03 查看预测结果

以下程序代码查看预测结果 DataFrame，如图 20-14 所示。

```
In [104]: predicted.select('url','features','rawprediction','probability','label','prediction').show(10)

+--------------------+--------------------+-------------+--------------------+-----+----------+
|                 url|            features|rawprediction|         probability|label|prediction|
+--------------------+--------------------+-------------+--------------------+-----+----------+
|http://1000awesom...|(36,[1,14,15,16,1...|  [20.0,143.0]|[0.12269938650306...|  1.0|       1.0|
|http://17andbakin...|(36,[2,14,15,16,1...|[118.0,378.0]|[0.23790322580645...|  1.0|       1.0|
|http://1x.com/pho...|(36,[0,15,16,17,2...|  [95.0,102.0]|[0.48223350253807...|  1.0|       1.0|
|http://2010.newsw...|(36,[2,14,15,16,1...|      [9.0,1.0]|           [0.9,0.1]|  0.0|       0.0|
|http://2oddities....|(36,[8,14,15,16,1...|     [27.0,0.0]|           [1.0,0.0]|  1.0|       0.0|
|http://3kidsandus...|(36,[0,15,16,17,1...|    [18.0,62.0]|       [0.225,0.775]|  0.0|       1.0|
|http://3kidsandus...|(36,[0,15,16,17,1...|    [37.0,63.0]|         [0.37,0.63]|  1.0|       1.0|
|http://6isx.com/i...|(36,[8,14,15,16,2...|     [15.0,8.0]|[0.65217391304347...|  0.0|       0.0|
|http://6jokes.com...|(36,[2,14,15,16,1...|     [49.0,2.0]|[0.96078431372549...|  0.0|       0.0|
|http://6jokes.com...|(36,[1,14,15,16,1...|     [20.0,2.0]|[0.90909090909090...|  0.0|       0.0|
+--------------------+--------------------+-------------+--------------------+-----+----------+
only showing top 10 rows
```

新增的字段

图 20-14 查看预测结果

新增的字段是：

- rawprediction：后续我们评估模型准确率时使用。

- prediction：预测的结果 0 或 1。
- probability：除了知道预测结果，还能够知道 0 或 1 的概率。

步骤04 查看预测结果与概率

上一步中 probability 字段太长，无法完全显示，可以使用 take(10) 查看。

```
In [105]: predicted.select('probability','prediction').take(10)

Out[105]: [Row(probability=DenseVector([0.1227, 0.8773]), prediction=1.0),
           Row(probability=DenseVector([0.2379, 0.7621]), prediction=1.0),
           Row(probability=DenseVector([0.4822, 0.5178]), prediction=1.0),
           Row(probability=DenseVector([0.9, 0.1]), prediction=0.0),
           Row(probability=DenseVector([1.0, 0.0]), prediction=0.0),
           Row(probability=DenseVector([0.225, 0.775]), prediction=1.0),
           Row(probability=DenseVector([0.37, 0.63]), prediction=1.0),
           Row(probability=DenseVector([0.6522, 0.3478]), prediction=0.0),
           Row(probability=DenseVector([0.9608, 0.0392]), prediction=0.0),
           Row(probability=DenseVector([0.9091, 0.0909]), prediction=0.0)]
```

我们可以从上面的运行结果看出 probability 是 DenseVector 格式：
[预测是 0 的概率，预测是 1 的概率]

- 例如，第 1 项数据 ([0.1227, 0.8773]), prediction=1.0)，[预测是 0 的概率为 0.1227，预测是 1 的概率为 0.8773]，因为预测是 1 的概率大于 0.5，所以最后预测结果是 1。
- 例如，第 4 项数据 ([0.9, 0.1]), prediction=0.0)，[预测是 0 的概率为 0.9，预测是 1 的概率为 0.1]，因为预测是 0 的概率大于 0.5，所以最后预测结果是 0。

20.6 评估模型的准确率

之前我们已经建立模型了，我们希望进一步评估模型的准确率。

步骤01 导入模块

首先从 pyspark.ml.evaluation 导入 BinaryClassificationEvaluator 模块。

```
In [107]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

步骤02 创建 BinaryClassificationEvaluator

创建 BinaryClassificationEvaluator，传入下列参数：

- rawPredictionCol="rawPrediction" 之前预测后产生的字段。

- labelCol="label" 标签字段。
- metricName="areaUnderROC" 也就是 AUC。

运行后创建 evaluator。

```
In [108]: evaluator = BinaryClassificationEvaluator(
                              rawPredictionCol="rawPrediction",
                              labelCol="label",
                              metricName="areaUnderROC"  )
```

步骤 03　计算AUC

可以执行下列命令查看 AUC。

```
In [243]: predictions =pipelineModel.transform(test_df)
          auc= evaluator.evaluate(predictions)
          auc
Out[243]: 0.6169097441250339
```

以上程序代码说明如下：

（1）使用 pipelineModel.transform 传入 test_df 测试数据进行预测，预测结果是 predictions。
（2）使用 evaluator.evaluate 传入 predictions 参数，计算 AUC。
（3）运行结果准确率是 0.61。

20.7 使用 TrainValidation 进行训练验证找出最佳模型

在第 13.10 节我们使用 Spark MLlib 必须自行编写 evalAllParameter 函数才能进行训练评估，以便找出最佳参数模型。然而，在 Spark.ML 中我们可以使用 TrainValidation 模块进行训练验证找出最佳模型。

步骤 01　导入模块

从 pyspark.ml.tuning 导入 ParamGridBuilder 与 TrainValidationSplit 模块。

```
In [56]: from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit
```

步骤 02　设置训练验证的参数

下面我们使用 ParamGridBuilder 设置 impurity 两个参数值、maxDepth 三个参数值与 maxBins 三个参数值，所以后续执行训练验证时会执行 2*3*3=18 次。

```
In [50]: paramGrid = ParamGridBuilder()\
          .addGrid(dt.impurity, [ "gini","entropy"])\
          .addGrid(dt.maxDepth, [ 5,10,15])\
          .addGrid(dt.maxBins, [10, 15,20])\
          .build()
```

步骤03 创建 TrainValidationSplit

创建 TrainValidationSplit，传入下列参数，执行后创建 tvs 变量：

- estimator=dt，之前创建的 DecisionTreeClassifier。
- evaluator=evaluator，之前创建的 BinaryClassificationEvaluator。
- estimatorParamMaps=paramGrid，之前创建的 ParamGridBuilder。
- trainRatio=0.8，训练验证前会先将数据按照 8 : 2 的比例分成训练数据与验证数据。

```
In [58]: tvs = TrainValidationSplit(estimator=dt,evaluator=evaluator,
                  estimatorParamMaps=paramGrid,trainRatio=0.8)
```

步骤04 建立 tvs_pipeline

建立 tvs_pipeline 阶段与之前建立的训练 pipeline 大致相同，只有最后一个阶段 tvs 是上一步骤所创建的 TrainValidationSplit，如图 20-15 所示。

```
In [59]: tvs_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, tvs])
```

只有最后一个阶段不同

图 20-15　建立 tvs_pipeline

步骤05. 使用 tvs_pipeline 流程进行训练验证

以下程序代码使用 tvs_pipeline.fit 进行训练与验证，传入 train_df 训练数据，训练数据会执行 tvs_pipeline 的所有阶段 stringIndexer、encoder、assembler 和 tvs，所以会比较费时，尤其是最后阶段（执行训练验证 18 次），最后产生的结果是 tvs_pipelineModel。

```
In [53]: tvs_pipelineModel =tvs_pipeline.fit(train_df)
```

步骤06 查看训练完成的最佳模型

tvs_pipelineModel 的第 4 阶段是 TrainValidation，因为是从 0 算起，所以 stages[3] 的 bestModel 属性是最佳模型。

```
In [279]: bestModel=tvs_pipelineModel.stages[3].bestModel
          bestModel
Out[279]: DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4a14b
          d409a76107a0437) of depth 25 with 1993 nodes
```



步骤07　查看训练验证完成的最佳模型规则

还可以用下列 toDebugString 来查看最佳模型规则，[:500] 只显示前 500 文字。

```
In [47]: print bestModel.toDebugString[:500]

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_
831b54624ffd91e93d7) of depth 15 with 1591 nodes
  If (feature 30 <= 1929.0)
   If (feature 2 in {1.0})
    If (feature 33 <= 0.284848485)
     If (feature 15 <= 0.373333333)
      If (feature 14 <= 3.405405405)
       If (feature 25 <= 0.018092105)
        If (feature 31 <= 36.0)
         Predict: 0.0
        Else (feature 31 > 36.0)
         If (feature 32 <= 7.0)
          Predict: 1.0
         Else (feature 32 > 7.0)
```

步骤08　评估最佳模型 AUC

最后以下列程序评估最佳模型 AUC 是 0.65。

```
In [253]: predictions = tvs_pipelineModel.transform(test_df)
          auc= evaluator.evaluate(predictions)
          auc
Out[253]: 0.6562313569532391
```

20.8　使用 crossValidation 交叉验证找出最佳模型

我们还可以更进一步用 crossValidation 交叉验证找出最佳模型。k-Fold 交叉验证（crossValidation）可以得到可靠稳定的模型，减少过度学习或学习不足的情况，一般常用为 10-Fold。k 值越大，效果越好，只是所需时间也越多。为了方便解说和避免执行时间过久，我们只以 k=3 来说明和示范。读者可以自行设置 k 值，只是需要考虑程序运行的机器性能，以免等候时间过久。

步骤01　crossValidation 交叉验证说明

k-Fold 的 crossValidation 交叉验证假设 k 是 3 的做法如下：



将数据分为 3 个部分 A、B、C，将会执行 3 次训练验证：

（1）B+C 作为训练数据，A 作为验证数据。

（2）A+B 作为训练数据，C 作为验证数据。

（3）A+C 作为训练数据，B 作为验证数据。

因为之前 ParamGridBuilder 设置 impurity 两个参数值、maxDepth 三个参数值与 maxBins 三个参数值，所以后续执行训练验证时会执行 2*3*3 = 18 次，所以总共会执行 18*3 = 54 次。

步骤02 导入模块

先导入模块，主要是从 pyspark.ml.tuning 导入 ParamGridBuilder 与 CrossValidation。

```
In [165]: from pyspark.ml.tuning import CrossValidator
```

步骤03 建立交叉验证的 CrossValidator

以下程序代码建立交叉验证的 CrossValidator 与之前创建 TrainValidationSplit 参数大致相同，只有最后一个参数 numFolds=3 不同。numFolds 的说明请参考步骤 1。

```
In [288]: cv = CrossValidator(estimator=dt, evaluator=evaluator,
                    estimatorParamMaps=paramGrid, numFolds=3)
```

步骤04 建立交叉验证的 cv_pipeline

建立 cv_pipeline 阶段与之前建立的训练 tvs_pipeline 大致相同，只有最后一个阶段 cv 是上一步骤所建立的 CrossValidator。

```
In [289]: cv_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, cv])
```

步骤05 使用 cv_pipeline 流程进行交叉验证

以下程序代码使用 cv_pipeline.fit 进行训练与验证，传入 train_df 训练数据。训练数据会执行 cv_pipeline 的所有阶段，尤其是最后阶段会执行训练验证 54 次，所以会比较费时，最后产生的结果是 cv_pipelineModel。

```
In [290]: cv_pipelineModel = cv_pipeline.fit(train_df)
```

步骤06 查看交叉验证完成的最佳模型

cv_pipelineModel 的第 4 阶段是 CrossValidator，因为是从 0 算起，所以 stages[3] 是 CrossValidator。CrossValidator 的 bestModel 属性是最佳模型。

```
In [285]: bestModel=cv_pipelineModel.stages[3].bestModel
          bestModel
Out[285]: DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4a14b
          d409a76107a0437) of depth 25 with 1993 nodes
```

步骤07 评估最佳模型 AUC

最后以下列程序评估最佳模型 AUC 是 0.65。

```
In [93]: predictions = cv_pipelineModel.transform(test_df)
         auc= evaluator.evaluate(predictions)
         auc
Out[93]: 0.6577640128548335
```

20.9 使用随机森林 RandomForestClassifier 分类器

随机森林的想法就是集合多棵决策树，每棵决策树都是分类器，都使用训练数据进行训练，如果有 *N* 棵树，就会产生 *N* 个分类结果。 每一棵树的分类结果可视为投票，随机森林整合了所有决策树的投票结果，将投票次数最多的类别作为最终的分类。使用机器学习流程的好处是，要更换组件很容易，我们只需要稍加修改之前决策树 DecisionTreeClassifier 的程序代码，就可以改成随机森林 RandomForestClassifier。

步骤 01 建立 RandomForestClassifier pipeline

以下程序代码建立 RandomForestClassifier pipeline 机器学习流程：

（1）首先导入 RandomForestClassifier 模块。

（2）创建 RandomForestClassifier 变量 rf，传入参数与决策树类似，只是多了 numTrees 参数（设置决策森林中有多少决策树，这里设为 10）。

（3）建立 rfpipeline，与之前决策树的程序代码大致相同，只有最后一阶段改为 rf。

```
In [96]: from pyspark.ml.classification import RandomForestClassifier

         rf =RandomForestClassifier(labelCol="label",
                        featuresCol="features",numTrees=10)

         rfpipeline = Pipeline(stages=[stringIndexer,encoder ,assembler,rf ])
```

步骤 02 评估 RandomForestClassifier 的准确度

rfpipeline.fit 传入 train_df 进行训练，再用 rftvs_pipelineModel.transform 传入 test_df 进行评估，我们可以看到 AUC 约为 0.73，比之前使用决策树的准确度明显增加。

```
In [69]: rfpipelineModel = rfpipeline.fit(train_df)
         rfpredicted=rfpipelineModel.transform(test_df)
         evaluator.evaluate(rfpredicted)
Out[69]: 0.7378858518228467
```

步骤 03 使用 RandomForestClassifier TrainValidation 找出最佳模型

以下 RandomForestClassifier 训练验证与 DecisionTree 大致相同，只是增加

了 .addGrid(rf.numTrees, [10, 20,30])。numTrees 用不同数值进行评估。

```
In [70]: from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
         from pyspark.ml.evaluation import BinaryClassificationEvaluator
         from pyspark.ml.classification import RandomForestClassifier

         paramGrid = ParamGridBuilder()\
          .addGrid(rf.impurity, [ "gini","entropy"])\
          .addGrid(rf.maxDepth, [ 5,10,15])\
          .addGrid(rf.maxBins, [10, 15,20])\
          .addGrid(rf.numTrees, [10, 20,30])\
          .build()

         rftvs = TrainValidationSplit(estimator=rf, evaluator=evaluator,
                          estimatorParamMaps=paramGrid, trainRatio=0.8)

         rftvs_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, rftvs])
         rftvs_pipelineModel =rftvs_pipeline.fit(train_df)
         rftvspredictions = rftvs_pipelineModel.transform(test_df)
         auc= evaluator.evaluate(rftvspredictions)
         auc

Out[70]: 0.7595738919870222
```

运行后结果 AUC 约为 0.759，准确度又增加了一些。

步骤04 使用 crossValidation 找出最佳模型

最后进行 crossValidation 交叉验证，与 DecisionTree 类似，只是原来的 dt 改为了 rf。

```
In [122]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
          from pyspark.ml import Pipeline

          rfcv = CrossValidator(estimator=rf, evaluator=evaluator,
                          estimatorParamMaps=paramGrid, numFolds=3)

          rfcv_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, rfcv])
          rfcv_pipelineModel = rfcv_pipeline.fit(train_df)
```

步骤05 使用最佳模型进行预测

使用 rfcvpredictions = rfcv_pipelineModel.transform(test_df)传入 test_df 进行预测。

```
In [88]: rfcvpredictions = rfcv_pipelineModel.transform(test_df)
```

步骤06 显示使用最佳模型预测结果

使用最佳模型预测后的结果如图 20-16 所示。

以上程序代码说明如下：

（1）创建 DescDict 字典，后续可用此字典转换预测结果的说明。

（2）for data in rfcvpredictions .select('url','prediction').take(5)用于读取前 5 项网址与预测结果。

（3）最后用 print 显示网址、预测结果、使用 DescDict 字典转换后的预测结果说明。

```
In [87]: DescDict = {
                0: "暂时性网页(ephemeral)",
                1: "长青网页(evergreen)"
            }
         for data in rfcvpredictions .select('url','prediction').take(5):
             print " 网址：  " +str(data[0])+"\n" +\
                   "         ==>预测:"+ str(data[1])+ \
                   " 说明:"+DescDict[data[1]] +"\n"

         网址：  http://1000awesomethings.com/2008/07/07/989-blowing-your-nose-in-the-shower/
                ==>预测:1.0 说明:长青网页(evergreen)

         网址：  http://24-7humor.com/what-the-hell-windows-95/
                ==>预测:0.0 说明:暂时性网页(ephemeral)

         网址：  http://2to5.list25.com/post/24398714566/worry-is-like-a-rocking-chair
                ==>预测:0.0 说明:暂时性网页(ephemeral)

         网址：  http://3kidsandus.com/2010/patriotic-checkerboard-cake-for-the-4th-of-july/
                ==>预测:1.0 说明:长青网页(evergreen)

         网址：  http://3kidsandus.com/red-velvet-rice-krispies-treats-hearts-for-valentines-day/
                ==>预测:1.0 说明:长青网页(evergreen)
```

图 20-16　显示使用最佳模型预测结果

步骤 07 计算最佳模型 AUC

用下列程序评估最佳模型 AUC 是 0.76。

```
In [89]: auc= evaluator.evaluate(rfcvpredictions)
         auc

Out[89]: 0.7616241115803595
```

运行后结果 AUC 约为 0.76，准确度又增加了一些。

20.10 结论

本章介绍了 Spark 机器学习流程（ML Pipeline）二元分类，包括建立 pipeline 机器学习流程，完成数据处理、训练模型、评估模型，并预测网页是暂时性还是长青的，最后使用训练验证与交叉验证找出最佳模型，提高预测准确度。在下一章，我们将介绍 Spark 机器学习流程（ML Pipeline）多元分类。

第 21 章

Spark ML Pipeline 机器学习流程多元分类

本章将使用第 17 章“森林覆盖树种”多元分类数据集来示范如何使用 Spark ML Pipeline 机器学习流程多元分类。使用决策树在不同条件（Elevation（海拔）、Aspect（方位）、Slope（斜率）、水源的垂直距离、荒野分类、水源的水平距离、土壤分类等等）预测 Cover Type 森林覆盖分类，并通过训练验证找出最佳模型，提高预测准确度。

➢ Spark Python 集成开发环境的安装步骤

“森林覆盖树种”数据集机器学习流程比较简单，只有两个阶段（见图 21-1）。

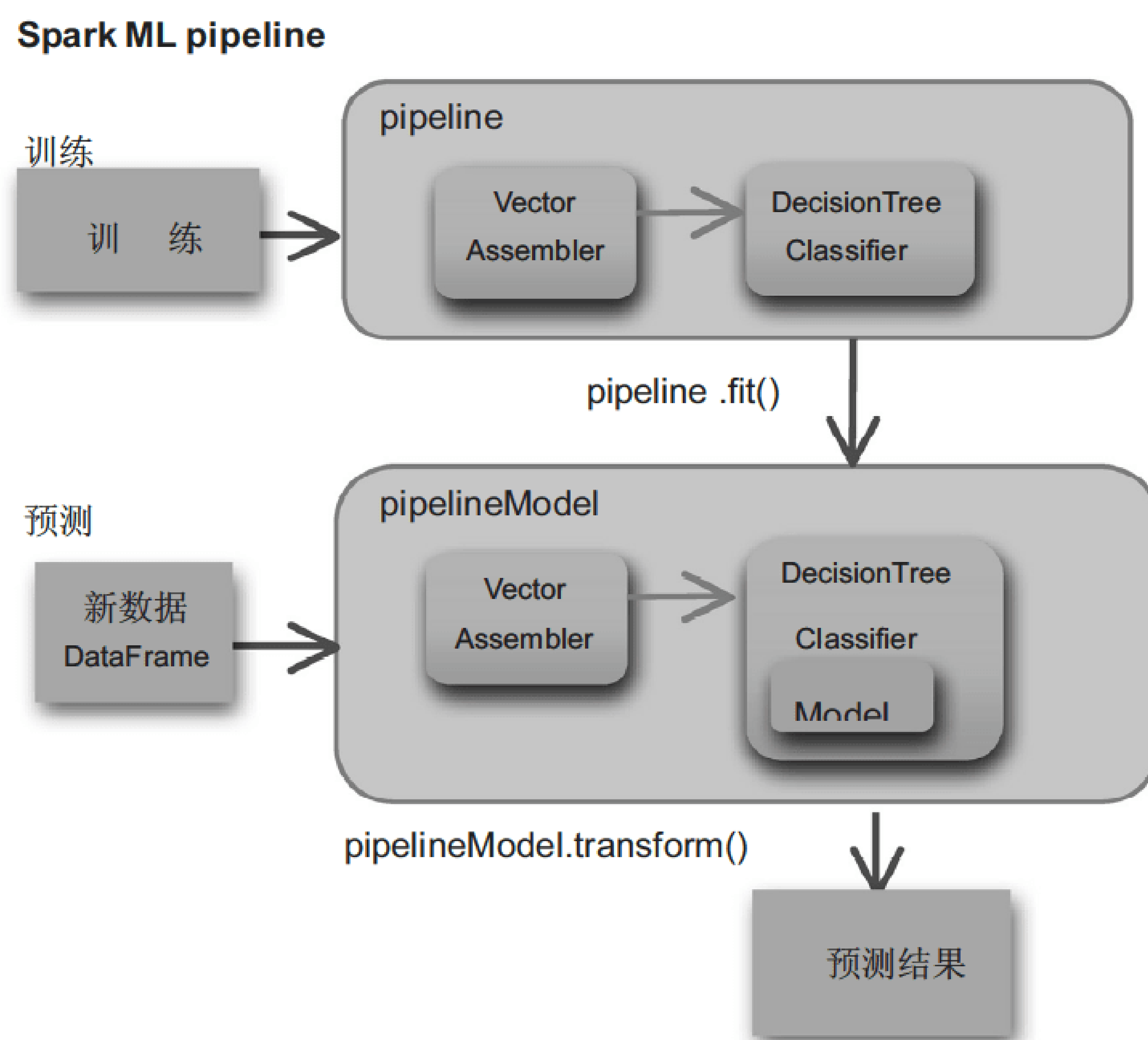


图 21-1　“森林覆盖树种”数据集机器学习流程

（1）**建立机器学习流程 pipeline：**包含两个阶段（stages），第一阶段是数据处理；第二阶段是 DesionTreeClassifier 机器学习多元分类算法。

- VectorAssembler：将所有的特征字段整合成 Vector。
- DesionTreeClassifier：决策树多元分类。

（2）**训练：**“训练数据 DataFrame”使用 pipeline .fit() 进行训练。系统会按照顺序执行每一个阶段，最后产生 pipelineModel 模型。pipelineModel 与 pipeline 类似，只是多了训练后建立的模型 Model。

（3）**预测：**“新数据 DataFrame”使用 pipelineModel.transform()。系统会按照顺序执行每一个阶段，最后使用 DesionTreeClassifier Model 进行预测。预测完成后会产生“预测结果 DataFrame”。

21.1 数据准备

我们将使用 IPython Notebook 来示范，因为使用 IPython Notebook 具有互动性的好处，可以看到指令执行后的结果。以下示范在本地执行，读者也可以参考第 9.9 节的说明，在不

同的模式运行 IPython Notebook。读者可以参考本书附录 A 有关本书范例程序下载与安装的说明，下载本章 IPython Notebook 范例文件进行练习。

➢ **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **本地执行 pyspark 使用 IPython Notebook 界面**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888，进去后就可以看到 IPython NoteBook 的界面。

21.1.1 读取文本文件

本章的示范将先以 sc.textFile 导入数据文件，然后再转换成 DataFrame。

步骤 01 配置文件的读取路径

在 IPython Notebook 使用下列指令配置文件的读取路径：

```
In [1]: global Path
        If sc.master[0:5]=="local" :
          Path="file:/home/hduser/pythonwork/PythonProject/"
        else:
          Path="hdfs://master:9000/user/hduser/"
```

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，代表当前是本地执行，读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，也就是在 cluster 执行，就有可能是 YARN client 或 Spark stand alone，读取 HDFS 文件。

步骤 02 读取文本文件并查看数据项数

以下程序代码先以 sc.textFile 导入数据文件，然后 rawData 使用 map 与 lambda 语句传入 x 作为参数，调用 x.split(",") 以逗号“,”分隔字段，提取每一字段。

```
In [3]: rawData = sc.textFile(Path+"data/covtype.data")
        lines = rawData.map(lambda x: x.split(","))
        lines.count()

Out[3]: 581012
```

以上运行结果显示共计 581012 项数据。

步骤 03 查看字段数

以 len() 查看数据集字段数共 55 个，并存放于 fieldnum 变量。

```
In [9]: fieldnum = len(lines.first())
        fieldnum
Out[9]: 55
```

21.1.2 创建 DataFrame

步骤01 导入 StringType 模块

首先，从 pyspark.sql.types 导入 StringType、StructField、StructType 模块，后续可用于创建字符串字段。

```
In [5]: from pyspark.sql.types import StringType,StructField,StructType
```

步骤02 创建 DataFrame

以下程序代码使用 for i in range(fieldnum) 语句从 i=0 到 54 共执行 55 次，用 StructField 创建字段，产生的字段名分别是 f0、f1、f2…f54，数据类型是 String，运行的结果存放在 fields 变量中。

```
In [208]: fields = [StructField("f"+str(i), StringType(), True)
                for i in range(fieldnum )]
```

步骤03 创建 schema

用 StructType() 传入 fields 作为参数创建 schema。

```
In [207]: schema = StructType(fields)
```

步骤04 创建 DataFrame

使用 spark.createDataFrame 创建 DataFrame 传入参数：lines RDD 与 schema。

```
In [8]: covtype_df = spark.createDataFrame(lines, schema)
```

步骤05 查看字段

以下列指令查看 covtype_df 字段，我们可以发现从 f0 到 f54 共 55 个字段。

```
In [106]: print covtype_df.columns
          ['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f
          9', 'f10', 'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17
          ', 'f18', 'f19', 'f20', 'f21', 'f22', 'f23', 'f24', 'f25'
          , 'f26', 'f27', 'f28', 'f29', 'f30', 'f31', 'f32', 'f33',
           'f34', 'f35', 'f36', 'f37', 'f38', 'f39', 'f40', 'f41',
          'f42', 'f43', 'f44', 'f45', 'f46', 'f47', 'f48', 'f49', '
          f50', 'f51', 'f52', 'f53', 'f54']
```

步骤06 查看 Schema

使用 printSchema() 来查看 covtype_df 的 Schema，我们可以看到所有字段都是 string

数据格式。因为字段太多，所以我们只截取了部分界面，如图 21-2 所示。

```
In [107]: print covtype_df.printSchema()

root
 |-- f0: string (nullable = true)
 |-- f1: string (nullable = true)
 |-- f2: string (nullable = true)
 |-- f3: string (nullable = true)
 |-- f4: string (nullable = true)
 |-- f5: string (nullable = true)
 |-- f6: string (nullable = true)
```

所有字段都是 string 数据格式

图 21-2　查看 Schema

21.1.3　转换为 double

必须先将所有字段转为 double，后续才能执行训练。

步骤 01　转换为 double

以下程序代码将所有字段转换为 double：

```
In [50]: from pyspark.sql.functions import col
         covtype_df= covtype_df.select([ col(column).cast("double").alias(column)
                          for column in covtype_df.columns])
```

以上指令说明如下：

（1）从 pyspark.sql.functions 导入 col 模块，后续我们可以用此模块读取字段数据。
（2）以 covtype_df.select 选取字段。
（3）for column in covtype_df.columns 执行所有字段。
（4）col(column)读取字段数据，并使用 .cast("double") 转换为 double。
（5）.alias(column)把别名设置为原来的字段名。

步骤 02　查看转换后的 Schema

使用 printSchema() 显示 Schema，转换后我们可以发现所有字段都已经转换为 double。（字段很多，下面截图只显示部分界面。）

```
In [111]: covtype_df.printSchema()

root
 |-- f0: double (nullable = true)
 |-- f1: double (nullable = true)
 |-- f2: double (nullable = true)
 |-- f3: double (nullable = true)
 |-- f4: double (nullable = true)
 |-- f5: double (nullable = true)
 |-- f6: double (nullable = true)
 |-- f7: double (nullable = true)
```

步骤 03　创建特征字段 List

特征字段是 f0~f53，所以使用 .columns[:54] 创建 featuresCols 特征字段 List。

```
In [108]: featuresCols=covtype_df.columns[:54]
          print featuresCols

['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f
9', 'f10', 'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17
', 'f18', 'f19', 'f20', 'f21', 'f22', 'f23', 'f24', 'f25'
, 'f26', 'f27', 'f28', 'f29', 'f30', 'f31', 'f32', 'f33',
 'f34', 'f35', 'f36', 'f37', 'f38', 'f39', 'f40', 'f41',
'f42', 'f43', 'f44', 'f45', 'f46', 'f47', 'f48', 'f49', '
f50', 'f51', 'f52', 'f53']
```

步骤 04　设置 label 字段

以下程序代码设置 label 字段：

```
In [ ]: covtype_df=covtype_df.withColumn("label", covtype_df["f54"] - 1).drop("f54")
```

（1）covtype_df 的第 54 个字段是 label，其值范围是 1~7（1—Spruce/Fir、2—Lodgepole Pine、3—Ponderosa Pine、4—Cottonwood/Willow、5—Aspen、6—Douglas-fir、7—Krummholz）。

（2）因为要执行 DecisionTreeClassifier 训练，所以规定 label 值一定是从 0 开始的数值。

（3）我们使用 .withColumn("label", covtype_df["f54"] - 1)创建 label 字段并且减 1，label 值范围为 0~6。

步骤 05　查看第一项数据

下列程序代码使用 covtype_df.show(1) 查看第一项数据：

```
In [89]: covtype_df.show(1)

+------+----+---+-----+---+-----+-----+-----+-----+------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-----+
|    f0|  f1| f2|   f3| f4|   f5|   f6|   f7|   f8|    f9|f10|f11|f12|f13|f14|f15|f16|f17|f18|f19|f20|f21|f22|f23|f24|f25|f26|f27|f28|
f29|f30|f31|f32|f33|f34|f35|f36|f37|f38|f39|f40|f41|f42|f43|f44|f45|f46|f47|f48|f49|f50|f51|f52|f53|label|
+------+----+---+-----+---+-----+-----+-----+-----+------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-----+
|2596.0|51.0|3.0|258.0|0.0|510.0|221.0|232.0|148.0|6279.0|1.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|1.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|  4.0|
+------+----+---+-----+---+-----+-----+-----+-----+------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-----+
only showing top 1 row
```

以上运行结果除了最后字段是 label 标签字段，其余都是 features 特征字段。

步骤 06　将数据分成 train_df 与 test_df

使用 randomSplit 将数据按照 7 : 3 的比例分成 train_df（训练数据）与 test_df（测试数据），并且 .cache() 暂存在内存中，以加快后续的程序运行速度。

```
In [112]: train_df, test_df = covtype_df.randomSplit([0.7, 0.3])
          train_df.cache()
          test_df.cache()

Out[112]: DataFrame[f0: double, f1: double, f2: double, f3: double, f4: double, f5: doub
le, f6: double, f7: double, f8: double, f9: double, f10: double, f11: double,
f12: double, f13: double, f14: double, f15: double, f16: double, f17: double,
f18: double, f19: double, f20: double, f21: double, f22: double, f23: double,
f24: double, f25: double, f26: double, f27: double, f28: double, f29: double,
f30: double, f31: double, f32: double, f33: double, f34: double, f35: double,
f36: double, f37: double, f38: double, f39: double, f40: double, f41: double,
f42: double, f43: double, f44: double, f45: double, f46: double, f47: double,
f48: double, f49: double, f50: double, f51: double, f52: double, f53: double,
label: double]
```

21.2 建立机器学习 pipeline 流程

接下来，建立“森林覆盖树种”数据集机器学习流程。

步骤01 导入模块

这里我们整理全部需要导入的模块。

```
In [43]: from pyspark.ml import Pipeline
         from pyspark.ml.feature import  VectorAssembler
         from pyspark.ml.classification import DecisionTreeClassifier
```

步骤02 建立 pipeline

以下程序代码建立 pipeline：

```
In [114]: vectorAssembler = VectorAssembler(inputCols=featuresCols,
                                            outputCol="features")
          dt = DecisionTreeClassifier(labelCol="label",featuresCol= 'features'
                                   ,maxDepth =5,maxBins=20)
          dt_pipeline = Pipeline(stages=[vectorAssembler ,dt])
```

以上程序代码说明如下：

（1）创建 VectorAssembler，将所有字段整合成一个 Vector 特征字段传入参数：

- inputCols=featuresCols，之前创建的 featuresCols 特征字段 List。
- outputCol="features"，运行后产生的 Vector 特征字段。

（2）创建 DecisionTreeClassifier，进行多元分类，传入参数：

- labelCol="label"，标签字段。
- featuresCol= "features"，前一阶段产生的 Vector 特征字段。
- maxDepth =5, maxBins=20，决策树训练参数。

（3）建立 dt_pipeline 机器学习流程，使用 Pipeline()传入 stages 阶段参数，stages=[vectorAssembler ,dt]。最后将运行结果存放在 dt_pipeline 变量中。

步骤03 查看 pipeline 阶段

创建 dt_pipeline 后，可以使用 getStages() 看到每一个阶段，在本范例只有两个阶段。

```
In [22]: dt_pipeline.getStages()
Out[22]: [VectorAssembler_4235a65d78f1d27870e6,
          DecisionTreeClassifier_4105b5875dac5aac84b2]
```



21.3 使用 dt_pipeline 进行数据处理与训练

之前我们已经建立了机器学习流程 dt_pipeline，下面使用 dt_pipeline 进行数据处理与训练。

步骤 01 使用 dt_pipeline.fit 进行训练

以下程序代码使用 dt_pipeline.fit 进行数据处理与训练，传入 train_df 训练数据。训练数据会执行 dt_pipeline 的所有阶段，最后产生的结果是 pipelineModel。

```
In [23]: pipelineModel=dt_pipeline.fit(train_df)
```

步骤 02 查看训练完成后的决策树模型

pipelineModel 的第 2 阶段会产生决策树的模型，因为从 0 算起，所以是 stages[1]，可以用下列指令查看训练完成后的决策树模型。

```
In [24]: pipelineModel.stages[1]
Out[24]: DecisionTreeClassificationModel (uid=DecisionTreeClassifier_
492184b14ab0d1130a89) of depth 5 with 63 nodes
```

步骤 03 查看训练完成后的决策树模型规则

还可以进一步使用 toDebugString 来查看决策树模型的规则，加上 [:500] 只显示前 500 文字。

```
In [25]: print pipelineModel.stages[1].toDebugString[:500]
DecisionTreeClassificationModel (uid=DecisionTreeClassifier
c5aac84b2) of depth 5 with 63 nodes
  If (feature 0 <= 3056.0)
   If (feature 0 <= 2582.0)
    If (feature 10 <= 0.0)
     If (feature 0 <= 2404.0)
      If (feature 3 <= 0.0)
       Predict: 3.0
      Else (feature 3 > 0.0)
       Predict: 2.0
```

21.4 使用 pipelineModel 进行预测

之前我们已经训练完成并建立模型，现在我们可以进行预测。

步骤01 使用 pipelineModel.transform 进行预测

以下程序代码 pipelineModel 使用 transform 方法传入 test_df 测试数据并进行预测。

```
In [26]: predicted = pipelineModel.transform(test_df)
```

步骤02 查看新增的字段

可以查看产生后的 Schema（参考图 21-3）。

```
In [72]: print predicted .columns

['f0', 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'f1
2', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f20', 'f21', 'f22', 'f23
', 'f24', 'f25', 'f26', 'f27', 'f28', 'f29', 'f30', 'f31', 'f32', 'f33', 'f34'
, 'f35', 'f36', 'f37', 'f38', 'f39', 'f40', 'f41', 'f42', 'f43', 'f44', 'f45',
 'f46', 'f47', 'f48', 'f49', 'f50', 'f51', 'f52', 'f53', 'label', 'features',
'rawPrediction', 'probability', 'prediction']
```

预测后新增的字段

图 21-3　查看新增的字段

步骤03 查看预测结果（见图 21-4）

预测后新增的字段

```
In [28]: predicted.select('features', 'rawPrediction', 'probability', 'label','prediction').show(10)

+--------------------+--------------------+--------------------+-----+----------+
|            features|       rawPrediction|         probability|label|prediction|
+--------------------+--------------------+--------------------+-----+----------+
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  2.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  2.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  5.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  2.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  5.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  2.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  5.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  5.0|       2.0|
|(54,[0,1,2,3,4,5,...|[0.0,453.0,12079....|[0.0,0.0241716023...|  5.0|       2.0|
|(54,[0,1,2,5,6,7,...|[0.0,51.0,467.0,7...|[0.0,0.0334426229...|  5.0|       3.0|
+--------------------+--------------------+--------------------+-----+----------+
only showing top 10 rows
```

图 21-4　查看预测结果

从以上运行结果可以看到新增的字段。

- rawprediction：后续我们评估模型准确率时使用。
- prediction：预测的结果为 0~6。
- probability：除了知道预测结果，还能够预测 0~6 的概率。

步骤04 查看预测结果与概率

probability 字段太长，无法完全显示，这里使用 take(10) 来查看部分数据。



```
predicted.select('probability', 'prediction').take(10)

[Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0),
 Row(probability=DenseVector([0.0, 0.0334, 0.3062, 0.4813, 0.0, 0.179, 0.0]), prediction=3.0)]
```

从上面的运行结果可以看到 probability 是 DenseVector 格式：

- 例如，第 1 项数据([0.0, 0.0242, 0.6445, 0.0602, 0.0, 0.2711, 0.0]), prediction=2.0)从 0 算起第 2 个字段的预测概率 0.6445 最高，所以最后预测结果是 2。
- 例如，第 10 项数据[0.0, 0.0334, 0.3062, 0.4813, 0.0, 0.179, 0.0]), prediction=3.0)]从 0 算起第 3 个字段的预测概率 0.4813 最高，所以最后预测结果是 3。

21.5 评估模型的准确率

之前我们已经建立了模型，我们希望能评估模型的准确率。

步骤 01 导入模块

首先，自 pyspark.ml.evaluation 导入 MulticlassClassificationEvaluator 模块。

```
In [ ]: from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

步骤 02 创建 MulticlassClassificationEvaluator

创建 MulticlassClassificationEvaluator 传入下列参数：

- labelCol="label"，标签字段。
- predictionCol="prediction"，预测字段。
- metricName="accuracy"，准确率。

运行后创建 evaluator。

```
In [169]: evaluator = MulticlassClassificationEvaluator(
                          labelCol="label", predictionCol="prediction",
                          metricName="accuracy")
```

步骤 03 计算 accuracy

可以执行下列命令计算 accuracy：

```
In [32]: predictions=pipelineModel.transform(test_df)
         accuracy = evaluator.evaluate(predictions)
         accuracy
Out[32]: 0.7032329245727552
```

以上程序代码说明如下：

（1）使用 pipelineModel.transform 传入 test_df 测试数据进行预测，预测结果是 predictions。
（2）使用 evaluator.evaluate 传入 predictions 参数，计算 accuracy。
（3）运行结果准确率大约是 0.7。

21.6 使用 TrainValidation 进行训练验证找出最佳模型

在第 13.10 节中，我们使用 Spark MLlib 必须自行编写 evalAllParameter 函数来进行训练评估，找出最佳参数模型。在 Spark.ML 中，我们可以使用 TrainValidation 模块进行训练验证来找出最佳模型。

步骤01 导入模块

首先，从 pyspark.ml.tuning 导入 ParamGridBuilder 与 TrainValidationSplit 模块。

```
In [56]: from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit
```

步骤02 设置训练验证的参数

我们使用 ParamGridBuilder 设置 impurity 两个参数值、maxDepth 三个参数值与 maxBins 三个参数值，所以后续执行训练验证时会执行 2*3*3 = 18 次。

```
In [79]: paramGrid = ParamGridBuilder()\
          .addGrid(dt.impurity, [ "gini","entropy"])\
          .addGrid(dt.maxDepth, [ 10,15,25])\
          .addGrid(dt.maxBins, [30,40,50])\
          .build()
```

步骤03 创建 TrainValidationSplit

创建 TrainValidationSplit 传入下列参数，运行结果是 tvs 变量：

- estimator=dt，之前创建的 DecisionTreeClassifier。
- evaluator=evaluator，之前创建的 MulticlassClassificationEvaluator。
- estimatorParamMaps=paramGrid，之前创建的 ParamGridBuilder。
- trainRatio=0.8，训练时会先将数据按照 8:2 的比例分成训练数据与验证数据。

```
In [ ]: tvs = TrainValidationSplit(estimator=dt,evaluator=evaluator,
                 estimatorParamMaps=paramGrid,trainRatio=0.8)
```

步骤 04 建立 tvs_pipeline

建立 tvs_pipeline ，包含阶段与之前建立的训练 pipeline 大致相同，只有最后一个阶段 tvs 是上一步所创建的 TrainValidationSplit。

```
In [183]: tvs_pipeline = Pipeline(stages=[vectorAssembler ,tvs])
```

只有最后一个阶段不同

步骤 05 使用 tvs_pipeline 流程进行训练验证

以下程序代码使用 tvs_pipeline.fit 进行训练与验证，传入 train_df 训练数据。训练数据会执行 tvs_pipeline 的所有阶段，最后阶段会执行训练验证 18 次，时间较长，最后产生的结果是 tvs_pipelineModel。

```
In [38]: tvs_pipelineModel =tvs_pipeline.fit(train_df)
```

步骤 06 查看训练完成的最佳模型

tvs_pipelineModel 的第 2 阶段是 TrainValidation，因为是从 0 算起，所以 stages[1] 的 bestModel 属性是最佳模型。可以用下列 toDebugString 来查看最佳模型规则，[:500] 只显示前 500 个文字。

```
In [39]: bestModel=tvs_pipelineModel.stages[1].bestModel
         print bestModel.toDebugString[:500]

DecisionTreeClassificationModel (uid=DecisionTreeClassifier
  If (feature 0 <= 2699.0)
   If (feature 0 <= 2454.0)
    If (feature 23 <= 0.0)
     If (feature 3 <= 0.0)
      If (feature 12 <= 0.0)
       If (feature 5 <= 661.0)
        If (feature 9 <= 1003.0)
         If (feature 5 <= 216.0)
          If (feature 6 <= 209.0)
           If (feature 9 <= 720.0)
            Predict: 2.0
```

步骤 07 使用最佳模型进行预测

先使用 predictions = tvs_pipelineModel.transform(test_df) 传入 test_df 进行预测，然后用 predictions 显示预测结果，并使用 .withColumn Renamed 将部分特征字段名称用中文来显示，如图 21-5 所示。

```
In [40]: predictions = tvs_pipelineModel.transform(test_df)
         result=predictions.withColumnRenamed("f0", "海拔") \
                               .withColumnRenamed("f1", "方位") \
                               .withColumnRenamed("f2", "斜率") \
                               .withColumnRenamed("f3", "垂直距离") \
                               .withColumnRenamed("f4", "水平距离") \
                               .withColumnRenamed("f5", "阴影")
         result.select("海拔","方位","斜率","垂直距离","水平距离","阴影","label","prediction").show(10)

         +------+-----+----+-----+----+-----+-----+----------+
         |    海拔|   方位|  斜率| 垂直距离 |水平距离|   阴影|label|prediction|
         +------+-----+----+-----+----+-----+-----+----------+
         |1859.0| 18.0|12.0| 67.0|11.0| 90.0|  2.0|       2.0|
         |1872.0| 27.0|16.0| 95.0|22.0|124.0|  2.0|       2.0|
         |1872.0| 35.0|21.0|120.0|18.0| 85.0|  5.0|       5.0|
         |1877.0| 19.0|18.0| 85.0|25.0|108.0|  2.0|       2.0|
         |1877.0| 27.0|24.0| 90.0|18.0| 95.0|  5.0|       5.0|
         |1879.0| 18.0|14.0|  0.0| 0.0|120.0|  5.0|       5.0|
         |1886.0| 16.0|15.0| 30.0| 7.0|150.0|  5.0|       5.0|
         |1889.0| 39.0|23.0|175.0|44.0|150.0|  5.0|       2.0|
         |1889.0|353.0|30.0| 95.0|39.0| 67.0|  5.0|       2.0|
         |1890.0| 34.0|22.0|162.0|40.0|180.0|  2.0|       2.0|
         +------+-----+----+-----+----+-----+-----+----------+
         only showing top 10 rows
```

图 21-5　使用最佳模型进行预测

步骤08 计算最佳模型 accuracy

以下列程序评估最佳模型 accuracy 是 0.93：

```
In [41]: accuracy = evaluator.evaluate(predictions)
         accuracy
Out[41]: 0.9300605670547402
```

21.7 结论

本章介绍了 Spark ML Pipeline 机器学习流程，包括建立 pipeline 机器学习流程，完成数据处理、训练模型、评估模型，并预测 Cover Type 森林覆盖分类，最后使用训练验证找出最佳模型，提高预测准确度。下一章，我们将介绍 Spark ML Pipeline 机器学习流程回归分析。

第 22 章

Spark ML Pipeline 机器学习流程回归分析

本章将使用第 18 章介绍的“Bike Sharing”数据集，示范如何使用 Spark 机器学习流程（ML Pipeline）回归分析。使用决策树回归分析，在不同的情况（季节、月份、时间、假日、星期、工作日、天气、温度、体感温度、湿度、风速等）下来预测每一小时的租用数量，并且使用训练验证与交叉验证找出最佳模型，提高预测准确度，最后介绍使用 GBT（Gradient-Boosted Trees）梯度提升决策树，进一步提高预测准确度。

➢ **“Bike Sharing”数据集机器学习流程（ML Pipeline）说明**

“Bike Sharing”（共享单车）数据集机器学习流程比较简单，只有三个阶段（见图 22-1）。

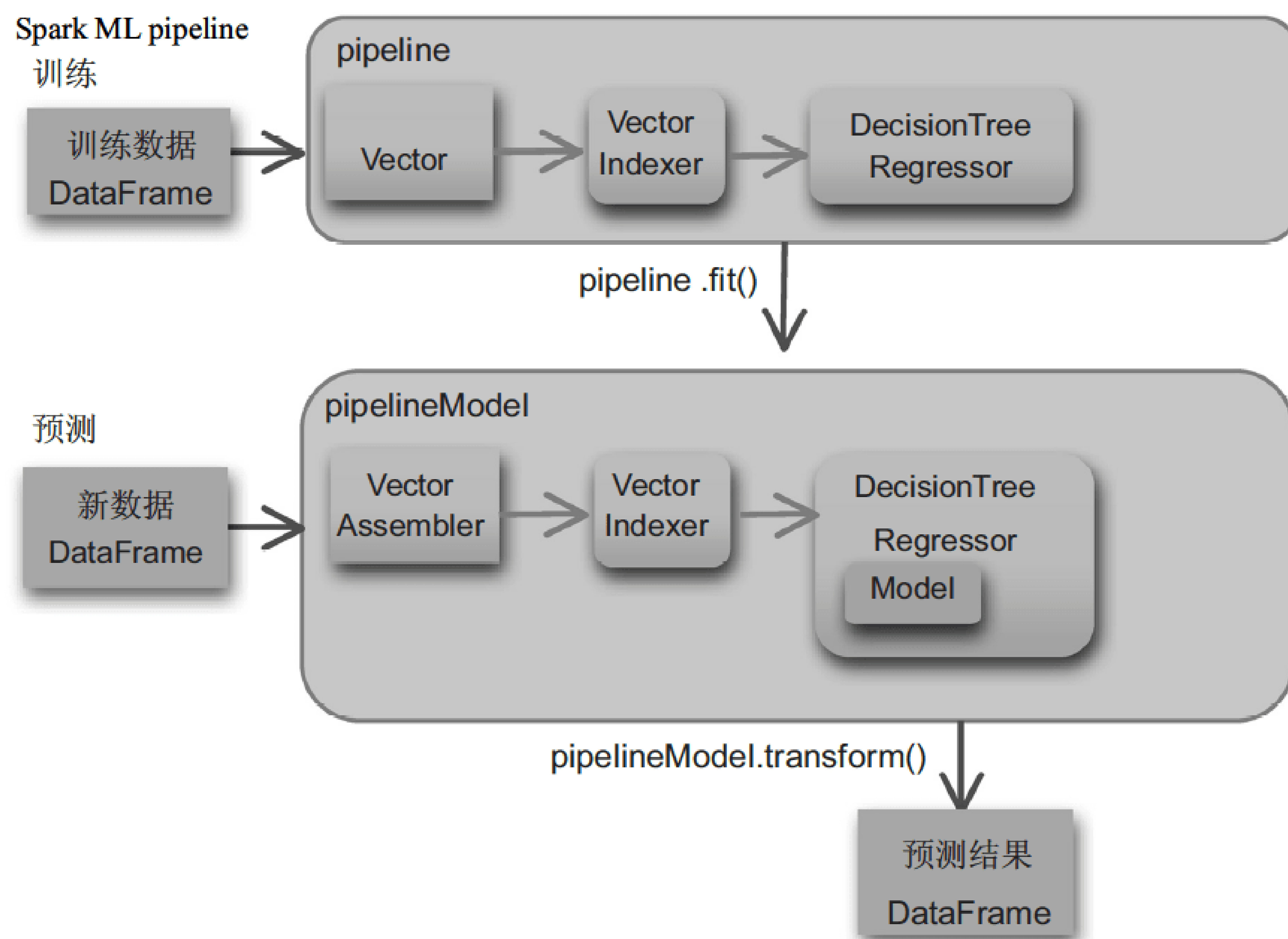


图 22-1　共享单车数据集机器学习流程

1. 建立机器学习流程 pipeline

包含 3 个阶段（stages），前 2 个阶段是数据处理，第 3 个阶段是 DecisionTreeRegressor 机器学习决策树回归分析。

- VectorAssembler：将所有的特征字段整合成 Vector。
- VectorIndexer：将不重复数值的数量小于等于 maxCategories 参数值所对应的字段视为分类字段，否则视为数值字段。
- DecisionTreeRegressor：决策树回归分析。

2. 训练

“训练数据 DataFrame”使用 pipeline .fit() 进行训练。系统会按照顺序执行每一个阶段，最后产生 pipelineModel 模型。pipelineModel 与 pipeline 类似，只是多了训练后建立的模型 Model。

3. 预测

“新数据 DataFrame”使用 pipelineModel.transform()。系统会按照顺序执行每一个阶段，最后使用 DecisionTree Regressor Model 进行预测。预测完成后会产生“预测结果 DataFrame”。

22.1 数据准备

我们将使用 IPython Notebook 示范，因为使用 IPython Notebook 具有互动性的好处，可以看到指令执行后的结果。以下示范在本地执行，读者也可以参考第 9.9 节的说明在不同的模式运行 IPython Notebook。读者可以参考本书附录 A 有关本书范例程序下载与安装的说明，下载本章 IPython Notebook 范例文件进行练习。

22.1.1 在 local 模式执行 IPython Notebook

在“终端”程序输入下列命令：

➢ **切换 ipynotebook 工作目录**

```
cd ~/pythonwork/ipynotebook
```

➢ **本地执行 pyspark 使用 IPython Notebook 界面**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark
```

按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888，进入后即可看到 IPython Notebook 的界面。

步骤 01 配置文件读取的路径

在 IPython Notebook 使用下列指令配置文件读取的路径：

```
In [1]: global Path
        if sc.master[0:5]=="local" :
          Path="file:/home/hduser/pythonwork/PythonProject/"
        else:
          Path="hdfs://master:9000/user/hduser/"
```

以上程序判断：

- 如果 sc.master[0:5] 是 "local"，代表当前是本地执行，读取本地文件。
- 如果 sc.master[0:5] 不是 "local"，也就是在 cluster 执行，就有可能是 YARN client 或 Spark stand alone，读取 HDFS 文件。

步骤 02 读取文本文件并且查看数据项数

下列指令使用 sqlContext.read 导入文本文件并创建 row_df DataFrame。

- .format("csv")，指定为 csv 格式。
- .option("header", "true")，指定第一行是字段名。

- .load(Path+"data/hour.csv")，指定要加载文件。
- 导入完成后，用 print row_df.count() 打印项数。

```
In [6]: hour_df= spark.read.format('csv') \
                   .option("header", 'true').load(Path+"data/hour.csv")
        hour_df.count()
Out[6]: 17379
```

以上运行结果显示共计17379 项数据。

步骤03 查看字段

可以使用 .columns 查看导入数据的所有字段。

```
In [60]: print hour_df.columns

         ['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', '
         weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'w
         indspeed', 'casual', 'registered', 'cnt']
```

步骤04 舍弃字段

参考第 18.3 节使用 .drop 舍弃不需要的字段。

```
In [34]: hour_df=hour_df.drop("instant").drop("dteday") \
                        .drop('yr').drop("casual").drop("registered")
```

步骤05 查看Schema

使用 printSchema() 来查看导入数据的 Schema，我们可以看到所有字段都是 string 数据格式，如图 22-2 所示。

```
In [11]: print hour_df.printSchema()

         root
          |-- season: string (nullable = true)
          |-- mnth: string (nullable = true)
          |-- hr: string (nullable = true)
          |-- holiday: string (nullable = true)
          |-- weekday: string (nullable = true)
          |-- workingday: string (nullable = true)
          |-- weathersit: string (nullable = true)
          |-- temp: string (nullable = true)
          |-- atemp: string (nullable = true)
          |-- hum: string (nullable = true)
          |-- windspeed: string (nullable = true)
          |-- cnt: string (nullable = true)
```

所有字段都是 string 数据格式

图 22-2 查看 Schema

步骤06 导入相关模块

我们需要从 pyspark.sql.functions 导入 col 模块，后续可以用此模块读取字段数据。



```
In [13]: from pyspark.sql.functions import col
```

步骤 07 将 string 转为 double

以下指令把 hour_df 转换为 double。

```
In [37]: hour_df= hour_df.select([ col(column).cast("double").alias(column)
                                   for column in hour_df.columns])
```

以上指令说明如下：

（1）用 hour_df.select 选取字段。
（2）for column in hour_df.columns 执行全部字段。
（3）col(column)读取字段数据。
（4）.cast("double") 转换为 double。
（5）.alias(column) 把别名设置为原来的字段名。

步骤 08 查看已经转换为 double（见图 22-3）

```
In [15]: hour_df.printSchema()

root
 |-- season: double (nullable = true)
 |-- mnth: double (nullable = true)
 |-- hr: double (nullable = true)
 |-- holiday: double (nullable = true)
 |-- weekday: double (nullable = true)
 |-- workingday: double (nullable = true)
 |-- weathersit: double (nullable = true)
 |-- temp: double (nullable = true)
 |-- atemp: double (nullable = true)
 |-- hum: double (nullable = true)
 |-- windspeed: double (nullable = true)
 |-- cnt: double (nullable = true)
```

已经转换为 double

图 22-3　查看已经转换为 double

步骤 09 查看结果

使用 hour_df.show(5) 查看结果：

```
In [39]: hour_df.show(5)

+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+
|season|mnth| hr|holiday|weekday|workingday|weathersit|temp| atemp| hum|windspeed| cnt|
+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+
|   1.0| 1.0|0.0|    0.0|    6.0|       0.0|       1.0|0.24|0.2879|0.81|      0.0|16.0|
|   1.0| 1.0|1.0|    0.0|    6.0|       0.0|       1.0|0.22|0.2727| 0.8|      0.0|40.0|
|   1.0| 1.0|2.0|    0.0|    6.0|       0.0|       1.0|0.22|0.2727| 0.8|      0.0|32.0|
|   1.0| 1.0|3.0|    0.0|    6.0|       0.0|       1.0|0.24|0.2879|0.75|      0.0|13.0|
|   1.0| 1.0|4.0|    0.0|    6.0|       0.0|       1.0|0.24|0.2879|0.75|      0.0| 1.0|
+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+
only showing top 5 rows
```

22.1.2 将数据分成 train_df 与 test_df

使用 randomSplit 将数据按照 7 : 3 的比例分成 train_df（训练数据）与 test_df（测试数据），并且 .cache() 暂存在内存中，以加快后续的程序运行速度。

```
In [40]: train_df, test_df = hour_df.randomSplit([0.7, 0.3])
         train_df.cache()
         test_df.cache()

Out[40]: DataFrame[season: double, mnth: double, hr: double, holiday: double
         , weekday: double, workingday: double, weathersit: double, temp: do
         uble, atemp: double, hum: double, windspeed: double, cnt: double]
```

22.2 建立机器学习 pipeline 流程

本章我们将建立如下机器学习 pipeline 流程：

（1）VectorAssembler：将多个特征字段整合成一个特征的 Vector。

（2）VectorIndexer：将不重复数值的数量小于等于 maxCategories 参数值所对应的字段视为分类字段，否则视为数值字段。

（3）DecisionTreeRegressor：决策树回归分析，预测每一小时租借数量。

为何要使用 VectorIndexer 呢？因为在 BikeSharing 数据集有月份 (1~12)、星期 (0~6)、小时 (0~23) 等字段。在决策树运算时，如果这些字段被视为分类字段，计算后准确率会比较高。

在本范例我们将设置 maxCategories = 24，如此月份（不重复数值的数量为 12）、星期（不重复数值的数量为 7）、小时（不重复数值的数量为 24）不重复数值的数量小于等于 maxCategories，所以月份、星期、小时都会被决策树算法视为分类字段。

如果我们设置 maxCategories = 4，如此月份、星期、小时都不会被决策树算法视为数值字段，因而计算后准确率会比较低。

步骤01 导入模块

以下导入模块与第 20 章大致相同，只是改为 VectorIndexer、DecisionTreeRegressor。

```
In [ ]: from pyspark.ml import Pipeline
        from pyspark.ml.feature import StringIndexer, VectorIndexer,VectorAssembler
        from pyspark.ml.regression import DecisionTreeRegressor
```

步骤02 创建特征字段 List

hour_df 除了最后一个字段是 label，其他都是特征字段，所以使用 hour_df.columns[:-1] 来获取特征字段 List。

```
In [42]: featuresCols = hour_df.columns[:-1]
         print featuresCols

['season', 'mnth', 'hr', 'holiday', 'weekday', 'workingday',
'weathersit', 'temp', 'atemp', 'hum', 'windspeed']
```

步骤 03 建立 pipeline

建立 pipeline 程序代码如下：

```
In [45]: vectorAssembler = VectorAssembler(inputCols=featuresCols, outputCol="aFeatures")
         vectorIndexer = VectorIndexer(inputCol="aFeatures", outputCol="features", maxCategories=24)
         dt = DecisionTreeRegressor(labelCol="cnt",featuresCol= 'features')
         dt_pipeline = Pipeline(stages=[vectorAssembler,vectorIndexer ,dt])
```

以上程序代码说明如下：

（1）创建 VectorAssembler 传入参数，结果存放于 vectorAssembler 变量：

- inputCols=featuresCols，之前步骤 2 时创建的特征字段 List。
- outputCol="aFeatures"，暂时输出的整合特征字段。

（2）创建 VectorIndexer 传入参数，结果存放于 vectorIndexer 变量：

- inputCol="aFeatures"，上一阶段 vectorAssembler 的输出字段。
- outputCol="features"，执行后产生的 features 特征字段。
- maxCategories = 24，月份、星期、小时都会被决策树算法视为分类字段。

（3）创建 DecisionTreeRegressor 传入参数，结果存放于 dt 变量：

- labelCol="cnt"，标签字段。
- featuresCol="features"，上一阶段产生的 features 特征字段。

最后，创建 dt_pipeline ： [vectorAssembler, vectorIndexer , dt]。

步骤 04 查看 pipeline 阶段

创建 dt_pipeline 后，可以使用 getStages() 看到每一个阶段。

```
In [44]: dt_pipeline .getStages()

Out[44]: [VectorAssembler_4a128d193260b8e52288,
          VectorIndexer_4c99ac7fd4f8d3692ab5,
          DecisionTreeRegressor_427db5d8359a7de978f1]
```

22.3 使用 dt_pipeline 进行数据处理与训练

之前我们已经建立机器学习流程 pipeline，下面使用 pipeline 进行数据处理与训练。

步骤 01 使用 dt_pipeline.fit 进行训练

以下程序代码使用 dt_pipeline.fit 进行数据处理与训练，传入 train_df 训练数据。

训练数据会执行 pipeline 的所有阶段 vectorAssembler、vectorIndexer 和 dt，最后产生的结果是 dt_pipelineModel。

```
In [ ]: dt_pipelineModel = dt_pipeline.fit(train_df)
```

步骤 02 查看训练完成后的决策树模型

pipelineModel 的第 3 阶段会产生决策树的模型，从 0 算起 stage[2]，可以用下列指令查看此模型。

```
In [52]: dt_pipelineModel.stages[2]
Out[52]: DecisionTreeRegressionModel (uid=DecisionTreeRegressor_
          of depth 5 with 63 nodes
```

步骤 03 查看训练完成后的决策树模型规则

还可以进一步使用 toDebugString 来查看决策树模型的规则。

```
In [47]: print dt_pipelineModel.stages[2].toDebugString[:500]

DecisionTreeRegressionModel (uid=DecisionTreeRegressor_427db5
d8359a7de978f1) of depth 5 with 63 nodes
  If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0,6.0,22.0,23.0})
   If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0})
    If (feature 2 in {1.0,2.0,3.0,4.0,5.0})
     If (feature 4 in {1.0,2.0,3.0,4.0,5.0})
      If (feature 2 in {2.0,3.0,4.0})
       Predict: 6.649106302916275
      Else (feature 2 not in {2.0,3.0,4.0})
       Predict: 21.048780487804876
     Else (feature 4 not in {1.0,2.0,3.0,4.0,5.0})
```

22.4 使用 pipelineModel 进行预测

之前我们已经训练完成并建立模型，现在我们可以进行预测。

步骤 01 使用 pipelineModel.transform 进行预测

以下程序代码 pipelineModel 使用 transform 方法传入 test_df 测试数据并进行预测。

```
In [55]: predicted_df=dt_pipelineModel.transform(test_df)
```

步骤 02 查看新增的字段

可以查看产生后的 Schema：

```
In [56]: print predicted_df.columns

['season', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersi
t', 'temp', 'atemp', 'hum', 'windspeed', 'cnt', 'aFeatures', 'features'
, 'prediction']
```

步骤 03 查看预测结果（见图 22-4）

```
In [63]: predicted_df.select('season', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', \
                  'weathersit', 'temp', 'atemp', 'hum', 'windspeed','cnt','prediction').show(10)

+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+-----------------+
|season|mnth| hr|holiday|weekday|workingday|weathersit|temp| atemp| hum|windspeed| cnt|       prediction|
+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+-----------------+
|   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.16|0.1364| 0.8|   0.2985|52.0|62.50292397660819|
|   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.26| 0.303|0.56|      0.0|39.0|62.50292397660819|
|   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.38|0.3939| 0.4|   0.2836|91.0|62.50292397660819|
|   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       1.0|0.06|0.0606|0.41|    0.194| 7.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       1.0|0.12|0.1212| 0.5|   0.2836| 5.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       1.0|0.24|0.2273| 0.6|   0.2239|15.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       2.0|0.24|0.2273|0.65|   0.2239| 7.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       1.0| 0.2| 0.197|0.51|   0.2537|13.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       2.0|0.26|0.2273| 0.7|   0.3284|12.0|38.00859598853868|
|   1.0| 1.0|0.0|    0.0|    3.0|       1.0|       1.0| 0.2|0.2576|0.64|      0.0| 6.0|38.00859598853868|
+------+----+---+-------+-------+----------+----------+----+------+----+---------+----+-----------------+
```

实际的数值　　预测的结果

图 22-4　查看预测结果

22.5 评估模型的准确率

之前我们已经建立了模型，下一步我们希望能评估模型的准确率。

步骤 01 导入模块

首先从 pyspark.ml.evaluation 导入 Regression Evaluator 模块。

```
In [64]: from pyspark.ml.evaluation import RegressionEvaluator
```

步骤 02 创建 Regression Evaluator

创建 Regression Evaluator 传入下列参数：

- labelCol='cnt'，标签字段。
- predictionCol='prediction'。
- metricName="rmse"，也就是 RMSE。

运行后创建 evaluator。

```
In [66]: evaluator = RegressionEvaluator(labelCol='cnt',
                                          predictionCol='prediction',
                                          metricName="rmse")
```

步骤03 计算 RMSE

可以执行下列命令来计算 RMSE：

```
In [67]: predicted_df=dt_pipelineModel.transform(test_df)
         rmse = evaluator.evaluate(predicted_df)
         rmse

Out[67]: 92.22331379768197
```

以上程序代码说明如下：

（1）使用 dt_pipelineModel.transform 传入 test_df 测试数据进行预测，预测结果是 predicted_df。

（2）使用 evaluator.evaluate 传入 predicted_df 参数，计算 RMSE。

（3）运行结果准确率是 92。

22.6 使用 TrainValidation 进行训练验证找出最佳模型

在第 13.10 节中，我们使用 Spark MLlib 必须自行编写 evalAllParameter 函数来进行训练评估，以便找出最佳参数模型。在 Spark.ML 中，我们可以使用 TrainValidation 进行训练验证，找出最佳模型。

步骤01 导入模块

先导入模块，主要是从 pyspark.ml.tuning 导入 ParamGridBuilder 与 TrainValidationSplit。

```
In [56]: from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit
```

步骤02 设置训练验证的参数

以下我们使用 ParamGridBuilder 设置 maxDepth 四个参数值与 maxBins 四个参数值，所

以后续执行训练验证时会执行 4*4 = 16 次。

请注意！因为我们设置了 maxCategories = 24，所以 maxBins 必须大于 24。

```
In [73]: paramGrid = ParamGridBuilder()\
           .addGrid(dt.maxDepth, [ 5,10,15,25])\
           .addGrid(dt.maxBins, [25,35,45,50])\
           .build()
```

步骤 03 创建 TrainValidationSplit

创建 TrainValidationSplit 传入下列参数，运行后创建 tvs。

- estimator = dt，之前创建的 DecisionTreeRegressor。
- evaluator = evaluator，之前创建的 RegressionEvaluator。
- estimatorParamMaps = paramGrid，之前创建的 ParamGridBuilder。
- trainRatio=0.8，训练时会先将数据按照 8 : 2 的比例分成训练数据与验证数据。

```
In [58]: tvs = TrainValidationSplit(estimator=dt,evaluator=evaluator,
                   estimatorParamMaps=paramGrid,trainRatio=0.8)
```

步骤 04 建立 tvs_pipeline

建立 tvs_pipeline 包含阶段与之前建立的训练 pipeline 大致相同，只有最后一个阶段 tvs 是上一步骤所创建的 TrainValidationSplit。

```
In [59]: tvs_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, tvs])
```

步骤 05 使用 tvs_pipeline 流程进行训练验证

以下程序代码使用 tvs_pipeline.fit 进行训练与验证，传入 train_df 训练数据。

训练数据会执行 tvs_pipeline 的所有阶段，最后阶段会执行训练验证 16 次，所以会比较费时，最后产生的结果是 tvs_pipelineModel。

```
In [58]: tvs_pipelineModel =tvs_pipeline.fit(train_df)
```

步骤 06 查看训练完成的最佳模型

tvs_pipelineModel 的第 3 阶段是 TrainValidation，因为是从 0 算起，所以 stages[2] 的 bestModel 属性是最佳模型。用下列 toDebugString 来查看最佳模型规则，[:500] 只显示前 500 个文字。

```
In [46]: bestModel=tvs_pipelineModel.stages[2].bestModel
         print bestModel.toDebugString[:500]

DecisionTreeRegressionModel
(uid=DecisionTreeRegressor_40d9a61275ccf736f9f3) of depth 10 with 1769
nodes
  If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0,6.0,22.0,23.0})
   If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0})
    If (feature 2 in {2.0,3.0,4.0,5.0})
     If (feature 4 in {1.0,2.0,3.0,4.0,5.0})
      If (feature 2 in {2.0,3.0,4.0})
       If (feature 2 in {3.0,4.0})
        If (feature 1 in {0.0,1.0,2.0,3.0,11.0})
         If (feature 7 <= 0.4)
          If (feature 0 in {0.0,1.0})
```

步骤07 评估最佳模型 RMSE

最后用下列程序评估最佳模型 RMSE 是 82。

```
In [78]: predictions = tvs_pipelineModel.transform(test_df)
         rmse= evaluator.evaluate(predictions)
         rmse

Out[78]: 82.82882328935622
```

22.7 使用 crossValidation 进行交叉验证找出最佳模型

我们还可以更进一步用 crossValidation 交叉验证找出最佳模型。k-Fold 交叉验证（crossValidation）可以得到可靠稳定的模型，能减少过度学习或学习不足的情况，一般常用为 10-Fold。k 值越大，效果越好，只是所需时间也越多。以下为了方便解说与避免执行时间过久，我们只以 k = 3 来说明示范。读者可以自行设置 k 值，只是需要考虑程序运行的机器性能，以免等候时间过久。

步骤01 crossValidation 交叉验证说明

k-Fold 的 crossValidation 交叉验证，假设 k 是 3 的做法如下：



将数据分为 3 个部分 A、B、C，会执行 3 次训练验证：

（1）B+C 作为训练数据，A 作为验证数据。

（2）A+B 作为训练数据，C 作为验证数据。

（3）A+C 作为训练数据，B 作为验证数据。

因为之前 ParamGridBuilder 设置了 maxDepth 四个参数值与 maxBins 四个参数值，所以后续执行训练验证时会执行 4*4 = 16 次，因此总共会执行 16*3 = 48 次。

步骤 02 导入模块

先导入模块，主要是从 pyspark.ml.tuning 导入 ParamGridBuilder 与 CrossValidation。

```
In [81]: from pyspark.ml.tuning import CrossValidator
```

步骤 03 创建交叉验证的 CrossValidator

以下程序代码创建交叉验证的 CrossValidator 与之前创建 TrainValidationSplit 的参数大致相同，但是增加了最后一个参数 numFolds = 3。

```
In [87]: cv = CrossValidator(estimator=dt, evaluator=evaluator,
                         estimatorParamMaps=paramGrid, numFolds=3)
```

步骤 04 建立交叉验证的 cv_pipeline

建立 cv_pipeline 包含阶段与之前建立的训练 tvs_pipeline 大致相同，只有最后一个阶段 cv 是上一步骤所创建的 CrossValidator。

```
In [88]: cv_pipeline = Pipeline(stages=[vectorAssembler,vectorIndexer ,cv])
```

步骤 05 使用 cv_pipeline 流程进行交叉验证

以下程序代码使用 cv_pipeline.fit 进行训练与验证，传入 train_df 训练数据。

训练数据会执行 cv_pipeline 的所有阶段，所以会比较费时，尤其是最后阶段，会执行训练验证 48 次，最后产生的结果是 cv_pipelineModel。

```
In [89]: cv_pipelineModel = cv_pipeline.fit(train_df)
```

步骤 06 评估最佳模型 RMSE

最后以下列程序评估最佳模型 RMSE 大约是 81。

```
In [65]: predictions = cv_pipelineModel.transform(test_df)
         rmse= evaluator.evaluate(predictions)
         rmse
Out[65]: 81.4868805893453
```

22.8 使用 GBT Regression

GBT（Gradient-Boosted Trees）或 GBDT（Gradient-Boosted Decision Trees）梯度提升决策树与随机森林 Random Forests 都是集合很多决策树，不同的是训练的方式。Random Forests

可以并行产生很多决策树，再整合了所有决策树的投票结果，将投票次数最多的类别作为最终的分类。GBT 一次只产生一棵决策树，再根据前一个决策树的结果决定如何产生下一个决策树，所以无法并行处理。

步骤01 创建 GBT Regression

以下程序代码创建 GBT Regression：

（1）首先导入 GBT Regression 模块。
（2）建立 GBT Regression 变量 gbt。
（3）建立 gbt_pipeline，与之前决策树的程序代码大致相同，只有最后一阶段改为 gbt。

```
In [40]: from pyspark.ml.regression import GBTRegressor
         gbt = GBTRegressor(labelCol="cnt",featuresCol= 'features')
         gbt_pipeline = Pipeline(stages=[vectorAssembler,vectorIndexer,gbt])
```

步骤02 使用 GBT Regression 流程训练评估

RMSE 越低，代表误差越小。以下程序代码执行训练、预测、计算 RMSE，结果 RMSE 约为 75，使用 GBT Regression 误差变小了。

```
In [41]: gbt_pipelineModel = gbt_pipeline.fit(train_df)
         predicted_df=gbt_pipelineModel.transform(test_df)
         rmse = evaluator.evaluate(predicted_df)
         rmse
Out[41]: 75.63126565293501
```

步骤03 使用 GBT Regression CrossValidation 找出最佳模型

以下设置 CrossValidator 找出最佳模型：

```
In [45]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
         from pyspark.ml.evaluation import RegressionEvaluator
         from pyspark.ml import Pipeline

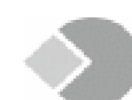
         paramGrid = ParamGridBuilder() \
           .addGrid(gbt.maxDepth, [ 5,10])\
           .addGrid(gbt.maxBins, [25,40])\
           .addGrid(gbt.maxIter, [10, 50])\
           .build()

         cv = CrossValidator(estimator=gbt, evaluator=evaluator,
                             estimatorParamMaps=paramGrid, numFolds=3)
         cv_pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])
```

步骤04 执行交叉验证

下列程序代码使用 cv_pipeline.fit 传入 train_df 训练数据，执行交叉验证。

```
In [46]: cv_pipelineModel = cv_pipeline.fit(train_df)
```

步骤 05　查看最佳模型

以下列程序代码查看最佳模型。

```
In [47]:  cvm=cv_pipelineModel.stages[2]
          gbestModel=cvm.bestModel
          print bestModel.toDebugString[:500]

          DecisionTreeRegressionModel (uid=DecisionTreeRegressor
            If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0,6.0,22.0,2
             If (feature 2 in {0.0,1.0,2.0,3.0,4.0,5.0})
              If (feature 2 in {2.0,3.0,4.0,5.0})
               If (feature 2 in {3.0,4.0})
                If (feature 4 in {1.0,2.0,3.0,4.0,5.0})
                 If (feature 1 in {0.0,1.0,2.0,3.0,11.0})
                  If (feature 7 <= 0.48)
                   If (feature 0 in {0.0,1.0})
                    If (feature 7 <= 0.4)
```

步骤 06　使用最佳模型进行预测

先使用 predicted_df = cv_pipelineModel.transform(test_df) 传入 test_df 进行预测，然后用 predicted_df 显示预测结果。

```
In [77]:  predicted_df=cv_pipelineModel.transform(test_df)
          predicted_df.select('season', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', \
                      'weathersit', 'temp', 'atemp', 'hum', 'windspeed','cnt','prediction').show(10)

          +------+----+---+-------+-------+----------+----------+----+------+----+---------+----+------------------+
          |season|mnth| hr|holiday|weekday|workingday|weathersit|temp| atemp| hum|windspeed| cnt|        prediction|
          +------+----+---+-------+-------+----------+----------+----+------+----+---------+----+------------------+
          |   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.04|0.0758|0.57|   0.1045|22.0| 42.77764555130705|
          |   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.16|0.1364|0.47|   0.3284|59.0|39.540499400708704|
          |   1.0| 1.0|0.0|    0.0|    0.0|       0.0|       1.0|0.16|0.1818| 0.8|   0.1045|33.0| 52.28270355182027|
          |   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       1.0|0.22| 0.197|0.44|   0.3582| 5.0|12.616150312553156|
          |   1.0| 1.0|0.0|    0.0|    1.0|       1.0|       2.0|0.32|0.2879|0.26|   0.4179|10.0|16.472294614491688|
          |   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       1.0|0.14|0.1667|0.59|   0.1045|12.0|11.369759093089817|
          |   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       2.0|0.22|0.2424|0.87|   0.1045|14.0|15.033981295186297|
          |   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       2.0|0.26|0.2273| 0.7|   0.3284|12.0|12.565447592592545|
          |   1.0| 1.0|0.0|    0.0|    2.0|       1.0|       2.0| 0.3|0.2879| 1.0|   0.2836|25.0| -6.54452578689676|
          |   1.0| 1.0|0.0|    0.0|    3.0|       1.0|       1.0| 0.2|0.2576|0.64|      0.0| 6.0|22.071060889516083|
          +------+----+---+-------+-------+----------+----------+----+------+----+---------+----+------------------+
          only showing top 10 rows
```

步骤 07　计算最佳模型 RMSE

用下列程序代码计算最佳模型 RMSE，结果 RMSE 约为 71，误差又小了一些。

```
In [78]:  evaluator = RegressionEvaluator(metricName="rmse",
                                          labelCol='cnt', predictionCol='prediction')
          rmse = evaluator.evaluate(predicted_df)
          rmse

Out[78]:  71.6723833260907
```

22.9　结论

本章介绍了 Spark 机器学习流程（ML Pipeline）多元分类，包括建立 pipeline 机器学习流程，完成数据处理、训练模型、评估模型，并预测每一小时租借总数量，最后使用训练验证与交叉验证找出最佳模型，提高预测准确度。

附录A
本书范例程序下载与安装说明

本书范例程序主要分为两个部分，说明如下：

范例	说明
IPython Notebook 范例	第 9、10、12、13、19、20、21、22 章，按照第 9 章的说明安装 anaconda 及其设置后才能使用这些范例程序
eclipse 项目范例	第 11~18 章，必须先按照第 11 章的说明完成 eclipse 的安装与全部设置后才能执行这些范例程序

A.1 下载范例程序

步骤 01 下载范例程序

启动 master 服务器的“终端”程序，输入下列指令：

➢ **切换到用户 home 目录**

```
cd ~/
```

如果登录的用户是 hduser，那么 home 目录就是 /home/hduser。

➢ **下载范例程序**

http://pan.baidu.com/s/1i4AzAk9

步骤 02 解压缩范例程序

在“终端”程序中输入下列命令来解压缩范例程序：

命令	说明
ll MP21622_example.zip	查看下载的程序
unzip MP21622_example.zip	解压缩范例程序

运行后屏幕显示界面如图 A-1 所示。

```
hduser@master: ~
hduser@master:~$ ll MP21622_example.zip
-rw-rw-r-- 1 hduser hduser 41123416  8月 31 15:49 MP21622_example.zip
hduser@master:~$ unzip MP21622_example.zip
```

图 A-1　解压缩范例文件

步骤 03 查看范例程序目录

执行解压缩后会解压缩在 ~/pythonsparkexample 目录中，可用以下命令查看：

命令	说明
cd ~/pythonsparkexample	切换范例目录
ll	查看目录

屏幕显示界面如图 A-2 所示。

```
hduser@master: ~/pythonsparkexample
hduser@master:~/pythonsparkexample$ ll
总用量 20
drwxrwxr-x  5 hduser hduser 4096  9月  5 10:42 ./
drwxr-xr-x 35 hduser hduser 4096  9月  5 10:23 ../
drwxrwxr-x  5 hduser hduser 4096  8月 30 02:30 ipynotebook/
drwxrwxr-x  3 hduser hduser 4096  8月 16 22:31 .metadata/
drwxrwxr-x  6 hduser hduser 4096  8月  4 11:14 PythonProject/
```

图 A-2　查看范例程序目录

以上目录说明如下：

目录	说明
ipynotebook/	第 9、10、12、13、19、20、21、22 章 IPython Notebook 范例
PythonProject/	第 11~18 章 eclipse 项目范例
.metadata/	eclipse 项目设置的相关文件，请不要删除

这些范例程序的说明请自行参考相关章节。

A.2 打开本书 IPython Notebook 范例程序

打开第 9、10、12、13、19、20、21、22 章 IPython Notebook 范例，先参照第 9 章的说明安装 anaconda 及其设置后才能使用这些范例程序。

步骤 01 启动 IPython Notebook

在“终端”程序中输入下列命令，进入 IPython Notebook 交互式界面。

- **切换 IPython Notebook 范例程序工作目录**

```
cd ~/pythonsparkexample/ipynotebook
```

- **运行 IPython Notebook 使用 Spark**

```
PYSPARK_DRIVER_PYTHON=ipython PYSPARK_DRIVER_PYTHON_OPTS="notebook" MASTER=local[4] pyspark
```



按 Enter 键后就会启动浏览器，默认的网址是 http://localhost:8888，进去后即可看到 IPython NoteBook 的界面，如图 A-3 所示。

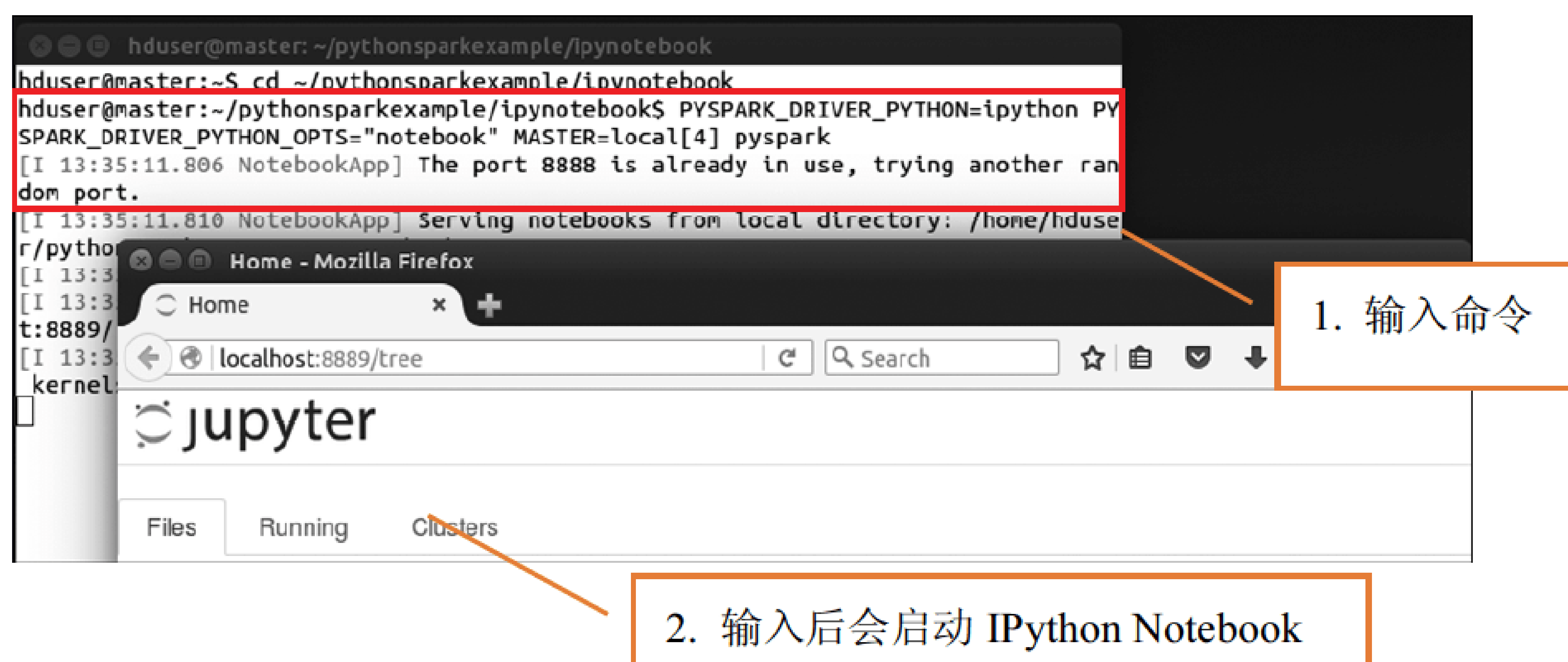


图 A-3 进入 IPython Notebook 界面

步骤 02 查看 IPython Notebook 范例程序（见图 A-4）

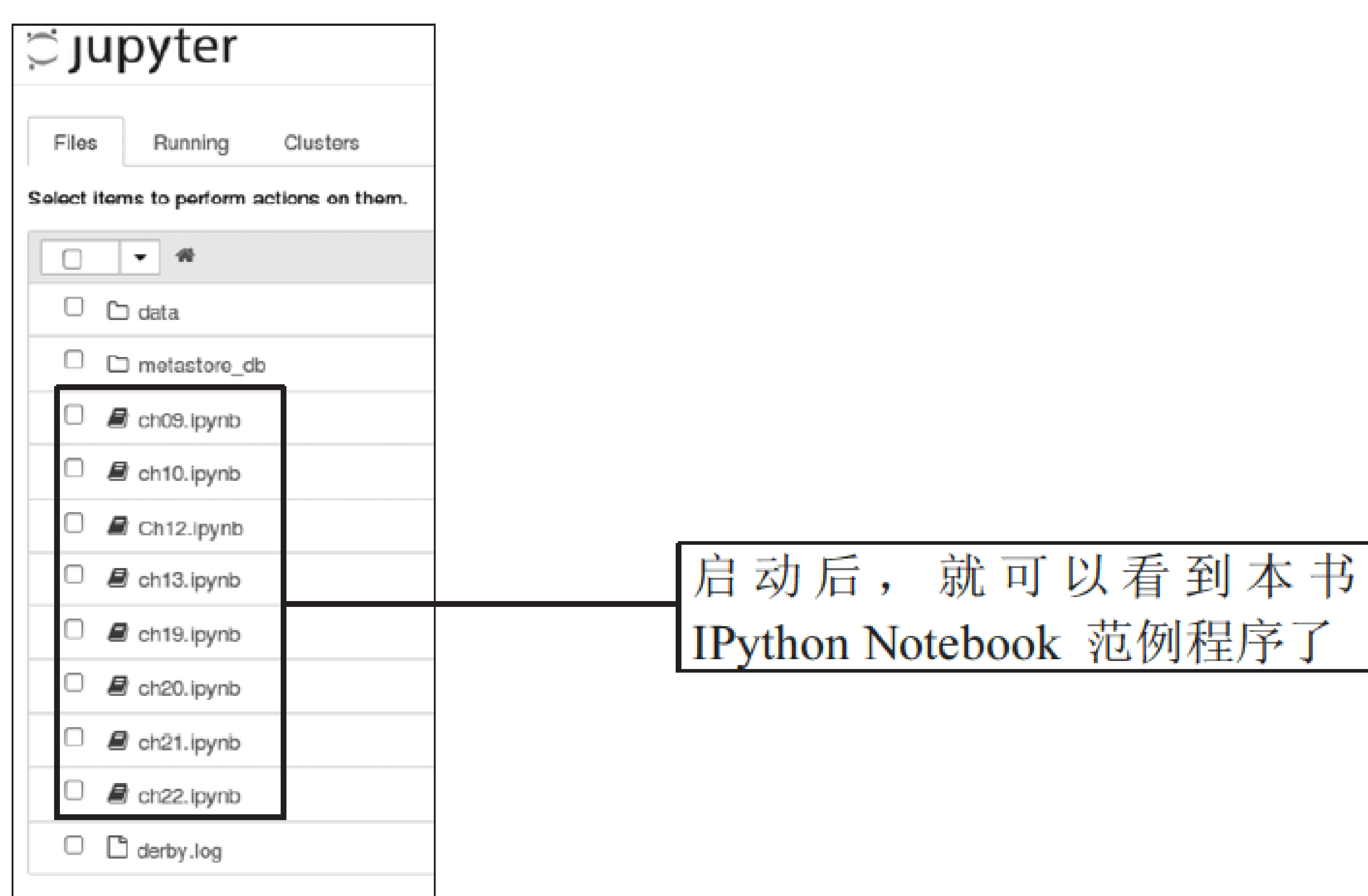


图 A-4 查看范例程序

步骤 03 本地文件的存取路径

为了能正确读取本地文件，所有范例程序在本地文件的存取路径都已经被修改为范例程序的路径 file:/home/hduser/pythonsparkexample/PythonProject/，如图 A-5 所示。

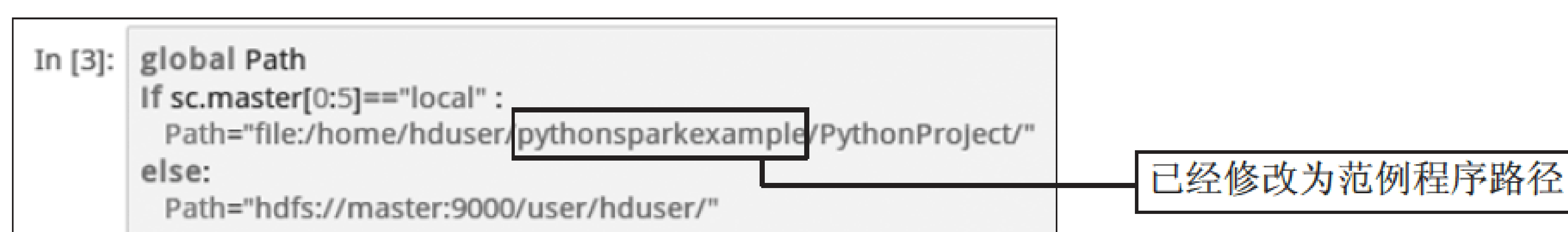


图 A-5 修改本地文件的存取路径

A.3 打开 eclipse PythonProject 范例程序

这是 eclipse 项目，所以必须先参照第 11 章的说明完成 eclipse 的安装与全部设置才能够执行这些范例程序。

步骤01 启动 eclipse

接下来，我们将使用 eclipse 打开 SparkExample 目录，如图 A-6 所示。



图 A-6 启动 eclipse

步骤02 设置工作目录（见图 A-7）

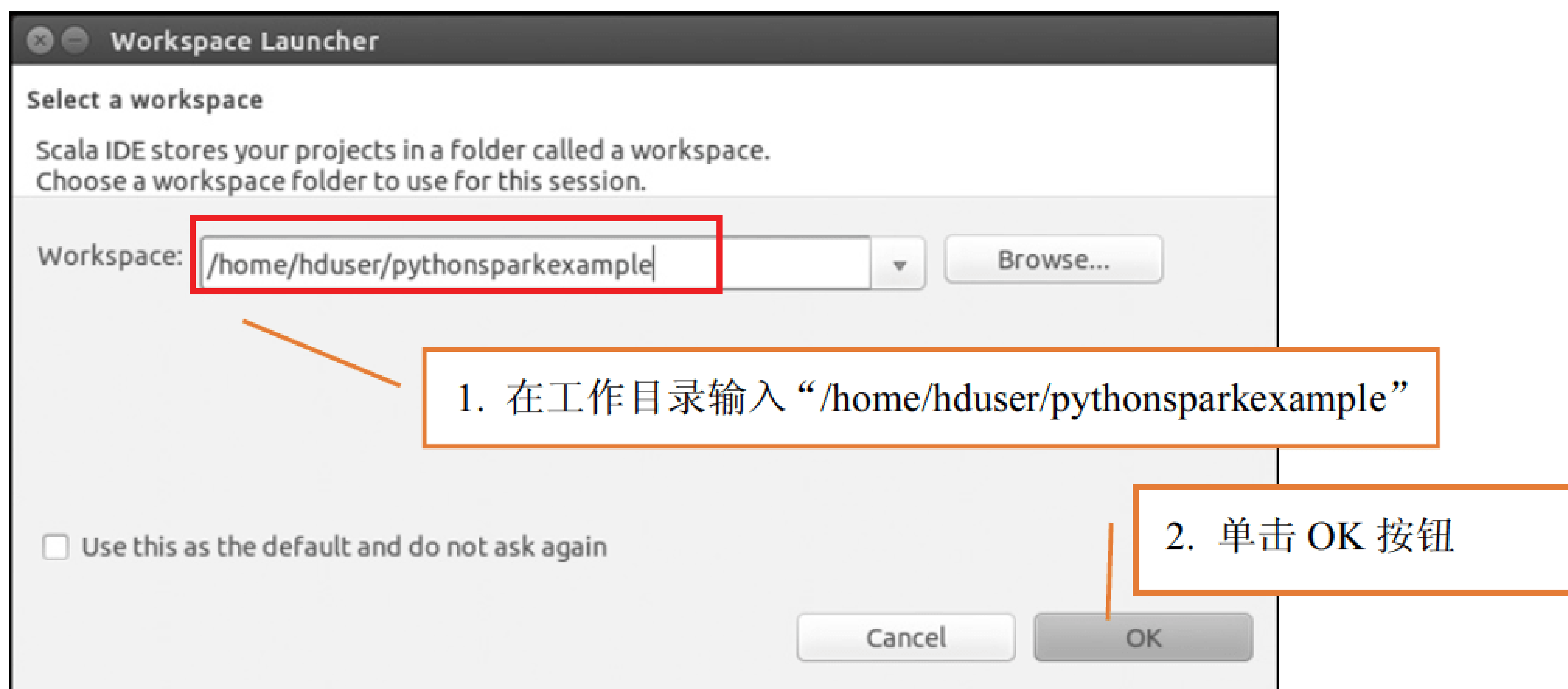


图 A-7 设置工作目录

步骤03 查看 PythonProject 项目

启动 eclipse 后，可以在 Package Explorer 看到 PythonProject 项目，第 11~18 章的相关范例程序就在其中，如图 A-8 所示。



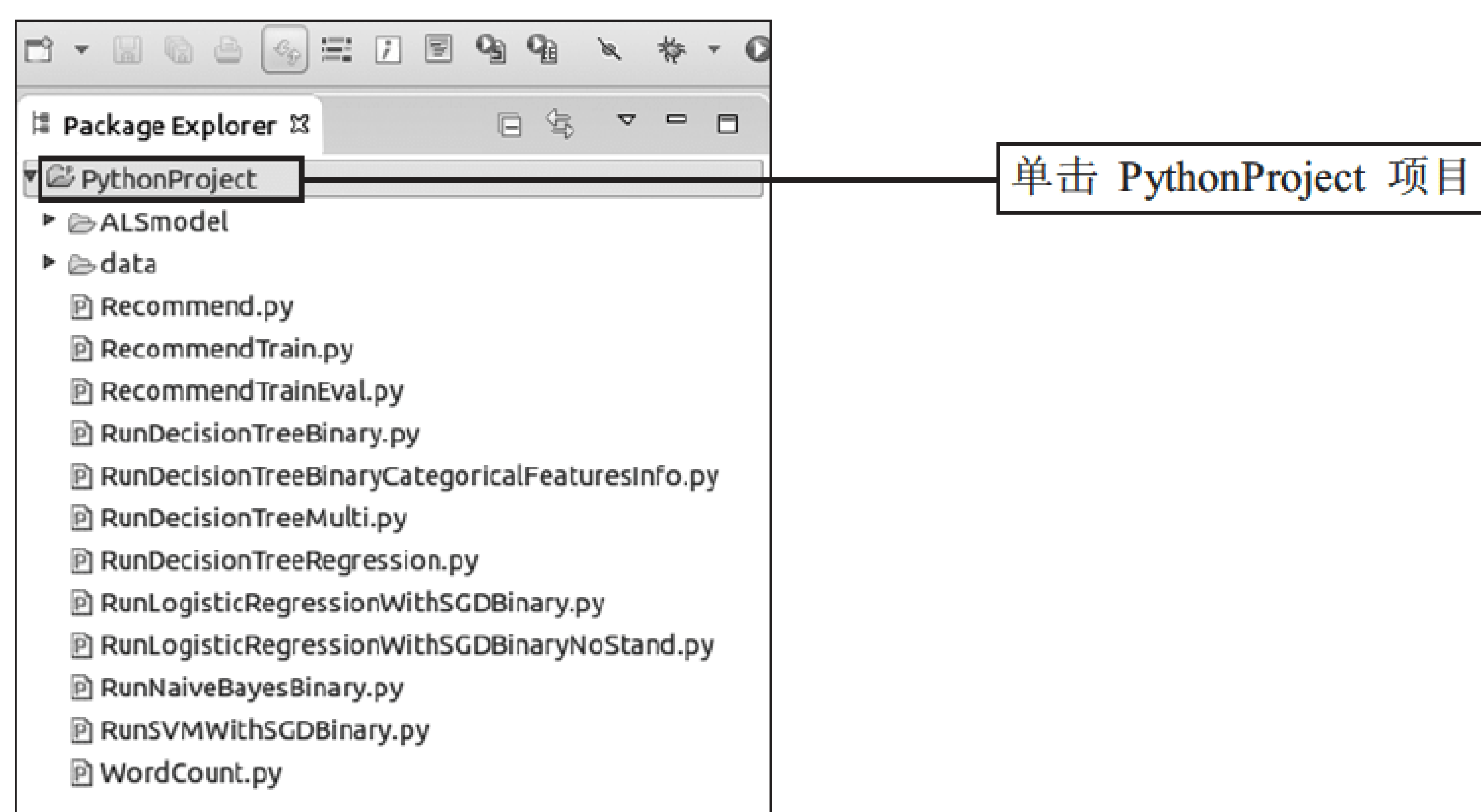


图 A-8　查看 Python Project 项目

步骤 04　本地文件的存取路径

为了能正确读取本地文件，所有的范例程序，本地文件的存取路径，我们都已经修改为范例程序路径 file:/home/hduser/pythonsparkexample/PythonProject/，如图 A-9 所示。

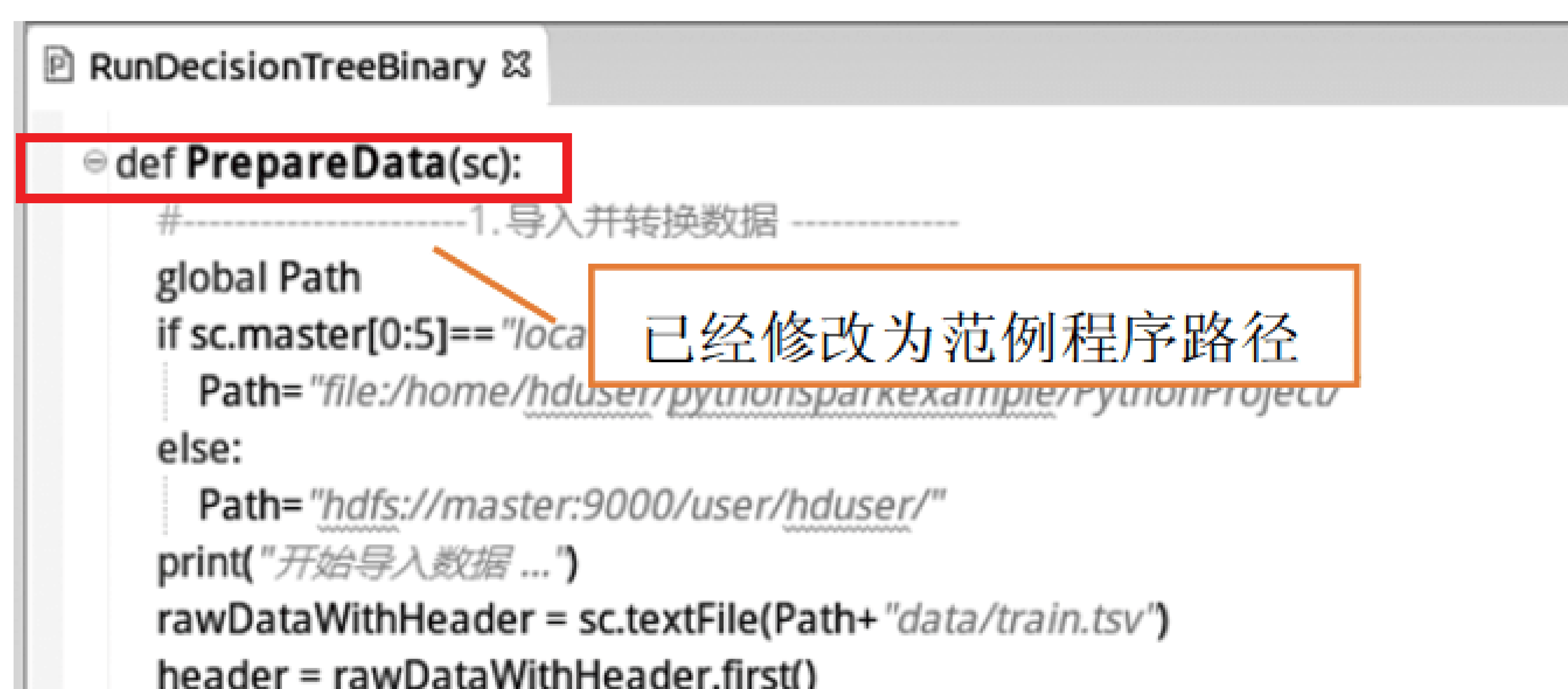


图 A-9　本地文件的存取路径